Journal of Computational Science xxx (2016) xxx–xxx

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

A randomized least squares solver for terabyte-sized dense overdetermined systems☆

Chander Iyer[a,*], Haim Avron[b], Georgios Kollias[c], Yves Ineichen[d], Christopher Carothers[a], Petros Drineas[a]

[a] Department of Computer Science, Rensselaer Polytechnic Institute, 110 8th Street, Troy, NY, USA
[b] Department of Applied Mathematics, Tel Aviv University, P.O. Box 39040, Tel Aviv, Israel
[c] IBM Research – T.J. Watson Research Center, Yorktown Heights, NY, USA
[d] IBM Research – Zurich Research Lab, Zurich, Switzerland

ARTICLE INFO

Article history:
Received 28 May 2016
Accepted 21 September 2016
Available online xxx

Keywords:
Randomized numerical linear algebra
High-performance computing
Dense least squares regression

ABSTRACT

We present a fast randomized least-squares solver for distributed-memory platforms. Our solver is based on the Blendenpik algorithm, but employs multiple random projection schemes to construct a sketch of the input matrix. These random projection sketching schemes, and in particular the use of the randomized Discrete Cosine Transform, enable our algorithm to scale the distributed memory vanilla implementation of Blendenpik to terabyte-sized matrices and provide up to ×7.5 speedup over a state-of-the-art scalable least-squares solver based on the classic QR algorithm. Experimental evaluations on terabyte scale matrices demonstrate excellent speedups on up to 16,384 cores on a Blue Gene/Q supercomputer.

1. Introduction

The explosive growth of data over the past 20 years in various domains, ranging from physics and biological sciences to economics and social sciences, has led to a need to perform efficient and scalable analysis on such massive datasets. One of the most widely and routinely used primitives in statistical data analysis is least-squares regression: given a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and a vector $\mathbf{b} \in \mathbb{R}^m$, we seek to compute

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbb{R}^n} \left\|\boldsymbol{A}\mathbf{x} - \mathbf{b}\right\|_2. \quad (1)$$

Several algorithms have been proposed to solve large-scale least-squares problems in various distributed and parallel environments [1], returning solutions whose accuracy is close to machine precision. Recent approaches for large-scale least-squares problems include communication-avoiding factorizations [2], which scale well for a variety of shared memory and distributed memory platforms [3] and are based on the classic QR decomposition algorithm, an $O(mn^2)$ algorithm (assuming $m \geq n$).

Recent years have witnessed an explosion of research on so-called Randomized Numerical Linear Algebra [4] (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 Eq. (1). Sarlos [5] and Drineas et al. [6] introduced the first randomized algorithms for this problem. These algorithms are based on the application of the sub-sampled Randomized Hadamard Transform to the 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 [7], who used a subsampled Randomized Fourier Transform to form a preconditioner and then used a standard iterative solver to solve the preconditioned problem. At the same time, Avron et al. [8] 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. [9] for a recent survey.

So far, most research on randomized least-squares regression algorithms has focused on the single processor setting, with two important exceptions. Meng et al. [10] introduced LSRN, a distributed memory algorithm for 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

☆ This work was partially supported by the XDATA program of the Defense Advanced Research Projects Agency (DARPA), administered through Air Force Research Laboratory contract FA8750- 12-C-0323, as well as by NSF IIS-1302231.
* Corresponding author.

http://dx.doi.org/10.1016/j.jocs.2016.09.007
1877-7503/

to both constants in the asymptotic analysis, as well as its much improved efficiency on parallel environments. Yang et al. [9] consider RandNLA in a MapReduce-like framework called Spark. This framework is less appropriate for supercomputers, as it is fails to take advantage of their hardware architecture.

In this work, we explore the behavior of Blendenpik-type algorithms in a distributed memory setting. We show that variants of Blendenpik that use various batchwise transformations to compute preconditioners lead to implementations that are not only faster than state-of-the-art implementations of baseline least-squares solvers, but are also able to scale to much larger matrix sizes. In particular, we show that a Blendenpik based algorithm can solve least-squares regression problems with terabyte-sized (and larger) input matrices. Our implementation and experiments were run on AMOS,[1] the high-performance Blue Gene/Q supercomputer system at Rensselaer. AMOS has five racks, 5120 nodes (81,920 cores), and 81,920 GB of main memory. AMOS has a peak performance of one PetaFLOP (10^{15} floating point operations per second), and a 5-D torus network with 2 GB/s of bandwidth per link and 512 GB/s to 1 TB/s 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 up to 1024 nodes (16,384 cores). The Blue Gene/Q architecture supports a hybrid communication framework that uses the MPI (Message Passing Interface) [10] standard for distributed communication and multithreading using OpenMP [12].

Our main contributions in this paper are: (i) implementation of and experimentation with the Blendenpik algorithm on distributed-memory platforms; (ii) implementation of four randomized sketching transforms in the context of the Blendenpik algorithm; (iii) a batchwise transformation scheme that scales a distributed vanilla implementation of the Randomized Discrete Cosine Transform in the context of the Blendenpik algorithm by up to three times in terms of matrix sizes, and provides a speedup of up to 7.5 times over a state-of-the-art scalable least-squares solver for tera-scale matrices; (iv) a detailed evaluation of four randomized sketching transforms in the context of the Blendenpik algorithm and their parameters on the BG/Q, using up to 16,384 cores.

The full source code of our batchwise Blendenpik implementation is available for download at https://github.com/cjiyer/libskylark/tree/batchwiseblendenpik. The rest of this paper is organized as follows. Section 2 describes the Blendenpik algorithm and the various stages of the algorithm in detail. Section 3 highlights the distributed Blendenpik implementation in the Blue Gene/Q, as well as scalability issues in our implementation, and describes an approach to overcome them. Section 4 first describes experiments to tune our Blue Gene/Q environment for our evaluations and then discusses the outcome of our experimental evaluations.

Notation. Let $\boldsymbol{A}$, $\boldsymbol{B}$, ... denote matrices and let $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$, ... 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 $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, let $\left\|\boldsymbol{A}\right\|_F^2 = \sum_{i,j=1}^{m,n} \boldsymbol{A}_{ij}^2$ 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 $\boldsymbol{A}$, where $r = rank(\boldsymbol{A})$ is the rank of the matrix $\boldsymbol{A}$. Then, the condition number of $\boldsymbol{A}$ is equal to $\kappa(\boldsymbol{A}) = \sigma_1/\sigma_r$.

2. The Blendenpik Algorithm for dense overdetermined systems

Blendenpik (see Algorithm 1) is a least-squares solver for dense, overdetermined, full column rank least-squares problems that computes an approximate solution to the problem of Eq. (1), with a high degree of precision. Given a dense, tall-and-thin matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and a column vector $\mathbf{b} \in \mathbb{R}^m$, the algorithm returns an approximate solution by executing the following three steps:

1. A preconditioner is constructed by applying a randomized unitary (or approximately unitary) transform $\boldsymbol{F}$ to the input matrix $\boldsymbol{A}$ and then sampling a small number of rows from the transformed matrix $\boldsymbol{FA}$ to form the matrix $\boldsymbol{M}_s$.
2. A QR factorization of the sampled matrix $\boldsymbol{M}_s$ is computed, returning an orthogonal matrix $\boldsymbol{Q}_s$ and an upper triangular matrix $\boldsymbol{R}_s$. The latter matrix $\boldsymbol{R}_s$ is then used as a preconditioner for the input matrix $\boldsymbol{A}$.
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 Eq. (1).

The algorithm uses a simple approach to estimate the condition number of the matrix $\boldsymbol{R}_s$. This procedure is described in [8] and amounts to computing the product $\left\|\boldsymbol{R}_s\right\|_1 \left\|\boldsymbol{R}_s^{-1}\right\|_1$, where $\left\|\boldsymbol{X}\right\|_1$ is the 1-norm of the matrix $\boldsymbol{X}$. 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 [8].

Input: $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ matrix, $m \gg n$, $rank(\boldsymbol{A}) = n$, $\mathbf{b} \in \mathbb{R}^m$, random transform matrix $\boldsymbol{F} \in \mathbb{R}^{m\times m}$, oversampling factor $\gamma \geq 1$, with $\gamma n \ll m$.
Output: approximate solution $\hat{\mathbf{x}}$ to the problem of Eq. (1).
while TRUE **do**
  Let $\boldsymbol{S} \in \mathbb{R}^{m\times m}$ be a random diagonal matrix:

$$\boldsymbol{S}_{ii} = \begin{cases} 1, & \text{with probability } \frac{\gamma n}{m} \\ 0, & \text{otherwise} \end{cases}$$

  $\boldsymbol{M}_s = \boldsymbol{SFA}$.
  Compute the QR factorization of $\boldsymbol{M}_s$: $\boldsymbol{M}_s = \boldsymbol{Q}_s\boldsymbol{R}_s$.
  Let $\hat{\kappa}$ be an estimate of the condition number of $\boldsymbol{R}_s$.
  if $\hat{\kappa}^{-1} > 5\epsilon_{\text{machine}}$ **then**
    $\mathbf{y} = \min_{\mathbf{z}} \left\|\boldsymbol{A}\boldsymbol{R}_s^{-1}\mathbf{z} - \mathbf{b}\right\|_2$.
    Solve $\boldsymbol{R}_s\hat{\mathbf{x}} = \mathbf{y}$ and **return** $\hat{\mathbf{x}}$.
  else if more than three iterations have been performed **then**
    Solve using the baseline least-squares solver and **return**.
  end if
end while

The most important stage of the Blendenpik algorithm is the application of the randomized transform $\boldsymbol{F}$; we will discuss various choices for $\boldsymbol{F}$ in Section 2.1. It is worth emphasizing that computing $\boldsymbol{M}_s$ as the product $\boldsymbol{SFA}$ in Algorithm 1 is for illustration purposes only. We will see in Section 2.1 that there are more efficient ways for computing $\boldsymbol{M}_s$ than simple matrix–matrix multiplication. The oversampling factor γ guarantees that as the number of rows of $\boldsymbol{M}_s$ will be (in expectation and with high probability) close to γn. As a result, the computation of the QR decomposition of $\boldsymbol{M}_s$ 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 $\boldsymbol{R}_s$ that is returned by the QR decomposition of $\boldsymbol{M}_s$ is then used as a preconditioner for the original problem. However, if $\boldsymbol{R}_s$ is ill-conditioned, then we repeat the generation of the randomized transform $\boldsymbol{F}$ in the hope of getting a better-conditioned matrix $\boldsymbol{R}_s$. If this procedure fails three times, then an exact solver is employed to solve the original least-squares problem. We conclude this discussion by stating that while setting γ to a smaller value

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

Please cite this article in press as: C. Iyer, et al., A randomized least squares solver for terabyte-sized dense overdetermined systems, J. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.jocs.2016.09.007

can improve the running time of the QR decomposition of $\boldsymbol{M}_s$, 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 [13] is an iterative, Krylov-subspace solver that works as follows: given the current iterate $\mathbf{x}_j$ and the corresponding residual error $\mathbf{r}_j = \mathbf{b} - \boldsymbol{A}\mathrm{x}_j$, LSQR uses the following criterion to test for convergence:

$$\frac{\left\| \left(\boldsymbol{A}\boldsymbol{R}_s^{-1}\right)^T \mathbf{r}_j \right\|_2}{\boldsymbol{A}\boldsymbol{R}_s^{-1} \left\| \mathbf{r}_j \right\|_2} \leq \rho,$$

where ρ is a tolerance value that determines the backward error at which the iterative solver terminates. This guarantees a backward stable solution to $\mathbf{y} = \min_{\mathbf{z}} \left\| \boldsymbol{A}\boldsymbol{R}_s^{-1}\mathbf{z} - \mathbf{b} \right\|_2$. 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 $\boldsymbol{A}\boldsymbol{R}_s^{-1}$ is, which in turn is determined by the oversampling factor γ.

2.1. The randomized transform $\boldsymbol{F}$

Our work is the first detailed evaluation on a Blue Gene/Q supercomputer of the Blendenpik algorithm with multiple choices for the randomized transform $\boldsymbol{F}$. We start by describing a few straightforward choices for $\boldsymbol{F}$, as well as more state-of-the-art choices for $\boldsymbol{F}$.

First, $\boldsymbol{F}$ could be a matrix whose entries are independent random variables, chosen from various well-known distributions. Perhaps the simplest choice is to set the entries of $\boldsymbol{F}$ to be independent Gaussian random variables of zero mean and variance $1/(\gamma n)$; we will refer to this construction of $\boldsymbol{F}$ as a Gaussian Random Transform (GRT). While Meng et al. [10] also use the GRT as their choice of randomized transform for the LSRN algorithm, a fundamental difference between LSRN and Blendenpik is the way the preconditioner is constructed. The Blendenpik algorithm uses a QR factorization of the sampled matrix $\boldsymbol{M}_s$ whereas the LSRN algorithm computes the SVD of $\boldsymbol{M}_s$ to form the preconditioner. A second straight-forward choice is to set the entries of $\boldsymbol{F}$ to be $+1/\sqrt{\gamma n}$ or $-1/\sqrt{\gamma n}$ with probability 1/2, independently for each entry. We will refer to this construction as a Random Sign Matrix (RSM). A third choice was proposed by Achlioptas in [14] and constructs the entries of $\boldsymbol{F}$ as follows:

$$\boldsymbol{F}_{ij} = \begin{cases} -\sqrt{3/(\gamma n)}, & \text{with probability } 1/6 \\ 0, & \text{with probability } 2/3 \\ +\sqrt{3/(\gamma n)}, & \text{with probability } 1/6 \end{cases}$$

Again, all entries are set to their respective values independently from all other entries. We refer to this construction as a Sparse Random Sign Matrix (SRSM). It is worth noting that all three aforementioned constructions, in the context of Blendenpik algorithm, would amount to first computing the product $\boldsymbol{S}\boldsymbol{F}$ (essentially only constructing the relevant rows of the randomized transform matrix, since rows that correspond to values $\boldsymbol{S}_{ii}$ equal to zero should not be constructed) and then applying $\boldsymbol{S}\boldsymbol{F}$ on to $\boldsymbol{A}$ by computing the product $(\boldsymbol{S}\boldsymbol{F})\boldsymbol{A}$. The SRSM transform could take advantage of the sparsity of the matrix $\boldsymbol{S}\boldsymbol{F}$ to compute the above product faster than the other two transforms. Finally, we conclude by noting that all three transforms are normalized appropriately in order to be approximately unitary.

Finally, our last random transform matrix $\boldsymbol{F}$ was proposed in the original Blendenpik paper [8]. The construction of $\boldsymbol{F}$ is the product of a random diagonal matrix and a fixed unitary transformation, namely the Discrete Cosine Transform (DCT). In this case, let $\boldsymbol{D}$ be a random diagonal matrix whose diagonal entries are set to +1 or −1 with probability 1/2. Then, let $\boldsymbol{C}$ be the matrix of the Discrete Cosine Transform (see [8] for details) and construct $\boldsymbol{F} = \boldsymbol{D}\boldsymbol{C}$. We will refer to this construction of $\boldsymbol{F}$ as the Randomized DCT (RDCT). We note that, in this case, the computation of $\boldsymbol{F}\boldsymbol{A}$ is more efficient that matrix-matrix multiplication. Indeed, one can apply the DCT matrix $\boldsymbol{C}$ on the columns of $\boldsymbol{A}$ in a column-wise manner much faster than matrix-vector multiplication, by using the properties of the Discrete Cosine Transform. Then, applying the matrices $\boldsymbol{S}$ and $\boldsymbol{D}$ on the resulting matrix $\boldsymbol{C}\boldsymbol{A}$ is trivial, since they are both diagonal matrices.

Table 1
Elemental data distribution overview.

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

3. Implementing our algorithm on the Blue Gene/Q

The algorithm is implemented on top of the Elemental library [15]. Given a distributed environment over p processes, any dense matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ 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 1 (not exhaustive). We use the standard distribution $[M_C, M_R]$ listed in Table 1 for dense input 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. Ref. [15] gives a comprehensive insight on these notations, describing different data distributions and the communication costs involved in redistribution.

We already presented in Section 2.1 various methods of generating the matrix $\boldsymbol{F}$ that is critical in the Blendenpik algorithm. When $\boldsymbol{F}$ is a Random Gaussian Transform, or a Random Sign Transform, or a Sparse Random Sign Transform, we generated and applied it within Elemental using the standard $[M_C, M_R]$ data distribution (a straight-forward operation). The only construction of $\boldsymbol{F}$ that merits additional discussion is the Randomized Discrete Cosine Transform. In order to apply the RDCT on a matrix $\boldsymbol{A}$ in a column-wise manner, we used the DCT implementation of FFTW [16], 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

Please cite this article in press as: C. Iyer, et al., A randomized least squares solver for terabyte-sized dense overdetermined systems, J. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.jocs.2016.09.007

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 1. In order to apply the DCT, all elements of a column must be stored locally, i.e., using the $[*, V_R/V_C]$ distribution. The Elemental pseudocode in order to apply the DCT in Blendenpik is described in the following steps:

$$\boldsymbol{FA} \Leftrightarrow \begin{cases} \boldsymbol{A}[\star, V_R] & = & \boldsymbol{A}[M_C, M_R] \\ M[\star, V_R] & = & \boldsymbol{F}[\star, V_R]\boldsymbol{A}[\star, V_R] \\ M[M_C, M_R] & = & M[\star, V_R] \end{cases}$$

The $\boldsymbol{A}[*, V_R] \longleftarrow \boldsymbol{A}[M_C, M_R]$ redistribution can be thought of as a `MPI_Scatter` and an `MPI_Gather` collective pair operation. The current MPI specifications (MPI 3.0) support sending and/or receiving up to `INT_MAX` ($2^{31}-1$) elements for any collective operation. This places memory constraints for terascale dense overdetermined systems, as the row sizes increase and more column elements get bunched together inside a single process. Hammond et al. [17] demonstrate a library implementation called BigMPI as a wrapper to current MPI specifications to resolve this problem. However, porting this wrapper to the local MPI implementation of our AMOS supercomputer is a cumbersome task and beyond the scope of our work. We instead overcame this problem using a batchwise unitary that we will discuss next.

Recall that the grid distribution of Elemental is cyclic and it therefore imposes an additional overload during redistribution of the input matrices. In more detail, the memory for each process is now shared by more than one columns of the input matrix. Since we are concerned with terascale, over-determined systems, a piecemeal redistribution, where in we select only a few columns to transform at a time is generally preferred. Another limiting factor that inhibits Blendenpik performance on terascale matrices is the communication cost in redistributing matrices after a batchwise transformation. A straightforward solution is to sample the batchwise transformed matrix before redistribution to generate the preconditioner. Let a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ be distributed in an $[M_C, M_R]$ format and let there be p MPI processes (for sake of this example) in our distributed environment. We divide the matrix $\boldsymbol{A}$ columnwise into b submatrices given by $\boldsymbol{A}^{(1)}, \boldsymbol{A}^{(2)}, \ldots, \boldsymbol{A}^{(b)}$, redistribute each submatrix $\boldsymbol{A}^{(i)}[\star, V_R] \leftarrow \boldsymbol{A}^{(i)}[M_C, M_R] : i \in \{1, \ldots, b\}$, perform random unitary transformations on each submatrix $M^{(i)}[\star, V_R] \leftarrow \boldsymbol{F}[\star, V_R]\boldsymbol{A}^{(i)}[\star, V_R] : i \in \{1, \ldots, b\}$, sample from each of the transformed submatrices $M_s^{(i)}[\star, V_R] \leftarrow \mathcal{S}[\star, V_R]M^{(i)}[\star, V_R] : i \in \{1, \ldots, b\}$, redistribute back each of the sampled b submatrices $M_s^{(i)}[M_C, M_R] \leftarrow M_s^{(i)}[\star, V_R] : i \in \{1, \ldots, b\}$, and finally merge them all in the $M_s[M_C, M_R]$ matrix. The number of columns in each submatrix and thus the number of submatrices b can be tuned depending on the dimensions of the matrix and the number of MPI processes p used in our evaluations.

4. Evaluation

To generate terascale-sized dense matrices, we randomly perturb sparse matrices with values generated from a standard normal distribution. We relied on the UFL Sparse matrix collection [18] for obtaining matrices of different condition numbers and coherence values. We chose to work with the Yoshiyasu Mesh matrix,[2] the ESOC Springer matrix,[3] and the Rucci[4] matrix from the UFL Sparse matrix collection for creating our datasets. Table 2 shows the particular matrices that were used in our evaluations: starting with the aforementioned three matrices, we first densified them by adding random Gaussian noise as discussed above. Then, we replicated and vertically concatenated them (with different noise added at each replication step) in order to create tall-and-thin matrices. Each matrix in Table 2 is suffixed with the number of replications and vertical concatenations of the base dense matrix. For example, to construct Yoshiyasu-24, we started with the Yoshiyasu Mesh matrix, added noise to create 24 slightly different copies of this matrix, and then vertically concatenated these 24 copies to get a tall-and-thin matrix.

We now describe our evaluation metrics. Recall that $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ is the input matrix and $\mathbf{b} \in \mathbb{R}^m$ is the 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 Eq. (1). Let $\hat{t}_{\text{run}}$ be the running time of Algorithm 1 and let t^*_{run} be the running time of the baseline Elemental least-squares solver. Then, our first accuracy metric is the relative error of the approximate solution $\hat{\mathbf{x}}$, given by the following formula:

$$\left\|\boldsymbol{A}\hat{\mathbf{x}} - \boldsymbol{A}\mathbf{x}^*\right\|_2 / \left\|\boldsymbol{A}\mathbf{x}^*\right\|_2. \tag{2}$$

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

$$\left\|\boldsymbol{A}^T\left(\mathbf{b} - \boldsymbol{A}\hat{\mathbf{x}}\right)\right\|_2. \tag{3}$$

Finally, the speedup of Algorithm 1 is defined as

$$t^*_{\text{run}}/\hat{t}_{\text{run}}. \tag{4}$$

4.1. AMOS environment setup

Our objective in the evaluation section is to provide a thorough evaluation of Blendenpik-type solvers for terascale least-squares problems from three primary viewpoints: scalability, performance, and accuracy. We tuned AMOS to evaluate the Blendenpik implementation against optimum baseline performance and scalability. We selected a combination of the number of OpenMP threads per node and the number of MPI processes per node that gave maximum performance for the baseline Elemental least-squares solver on AMOS. We measured the average time taken by the baseline solver over five runs for all possible MPI and OpenMP combinations for two dense matrices: a random matrix whose entries were generated uniformly at random in the interval $[-1,1]$ with dimensions $1{,}320{,}000 \times 38{,}000$ and the ESOC-4 matrix (Table 2) with dimensions $1{,}308{,}248 \times 37{,}830$. The AMOS system can execute up to 64 MPI processes with one OpenMP thread, or one MPI process with 64 OpenMP threads, or any combination of the two that results in a product of MPI processes and OpenMP threads that is equal to 64 in a single node. We used a `bgclang/LLVM` build of the baseline Elemental solver for our evaluations.

As seen in Table 3, the performance of the baseline Elemental solver improved as the number of OpenMP threads per node increased for a single MPI process per node. Similarly, the performance also increased with increasing MPI processes per node for a single OpenMP thread. However, the performance dropped when many MPI processes compete for CPU resources with several OpenMP threads in the system. The maximum performance is realized for 32 OpenMP threads for a single MPI process (Table 3), while it immediately degraded when the number of threads reached 64. One possible reason could be because of possible thread synchronization during floating point operations in the BG/Q CPU cores. The choice of one MPI process and 32 OpenMP threads per Blue Gene/Q node was the standard configuration that we selected for our evaluations.

[2] http://www.cise.ufl.edu/research/sparse/matrices/Yoshiyasu/mesh_deform.html.
[3] http://www.cise.ufl.edu/research/sparse/matrices/Springer/ESOC.html.
[4] http://www.cise.ufl.edu/research/sparse/matrices/Rucci/Rucci1.html.

Please cite this article in press as: C. Iyer, et al., A randomized least squares solver for terabyte-sized dense overdetermined systems, J. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.jocs.2016.09.007

Table 2
Matrices used in our evaluations.

Matrix name	# of rows (Millions)	# of columns	# of entries (Billions)	Size (TBs)
Yoshiyasu-1	0.234		2.198	0.016
Yoshiyasu-24	5.616		52.756	0.384
Yoshiyasu-48	11.233		105.512	0.768
Yoshiyasu-72	16.849		158.268	1.152
Yoshiyasu-96	22.466	9393	211.025	1.535
Yoshiyasu-120	28.082		263.781	1.919
Yoshiyasu-144	33.699		316.537	2.303
Yoshiyasu-168	39.315		369.294	2.687
Yoshiyasu-192	44.932		422.050	3.070
ESOC-1	0.327		12.373	0.090
ESOC-8	2.616		98.982	0.720
ESOC-16	5.233		197.964	1.440
ESOC-24	7.849		296.946	2.160
ESOC-32	10.466	37,830	395.928	2.880
ESOC-40	13.082		494.910	3.600
ESOC-48	15.698		593.892	4.321
ESOC-56	18.315		692.874	5.041
ESOC-64	20.931		791.856	5.761
Rucci-1	1.978		217.369	1.581
Rucci-2	3.956	109,900	434.739	3.163
Rucci-3	5.933		652.108	4.744

Table 3
Runtime analysis to select an optimal hybrid MPI/OpenMP configuration for our scaling experiments. Each run was averaged over five times on 128 BG/Q nodes. All runtimes are in seconds(lower runtimes are preferred); the optimal performance (bold running times) was observed when one MPI process per node and 32 OpenMP threads per node were used.

OpenMP threads per node	random matrix $\boldsymbol{A}$, of dimensions $1.32M \times 38K$; average $\kappa(\boldsymbol{A}) = 1.4084 \pm 1.1 * 10^{-4}$						
	MPI processes per node						
	1	2	4	8	16	32	64
1	8734.48	4613.62	2996.67	2187.48	1477.89	1878.74	1755.57
2	4499.6	2637.91	1762.49	1580.48	1240.34	1826.52	–
4	2349.32	1565.67	1156.55	1334.04	1168.86	–	–
8	1276.73	1097.84	922.341	2261.12	–	–	–
16	800.443	1057.67	1193	–	–	–	–
32	**628.171**	1081.64	–	–	–	–	–
64	686.517	–	–	–	–	–	–
OpenMP threads per node	ESOC-4 matrix $\boldsymbol{A}$, of dimensions $1.3082M \times 38K$; average $\kappa(A) = 1.4594 * 10^6 \pm 11.0268$						
	MPI processes per node						
	1	2	4	8	16	32	64
1	9304.71	5205.42	2737.43	2067.88	1378.84	1809.77	–
2	4831.84	2922.68	1640.01	1542.24	1133.12	1741.58	–
4	2567.38	1741.82	1091.47	1236.9	1023.93	–	–
8	1463.21	1207.34	898.165	1219.18	–	–	–
16	961.26	1026.97	891.234	–	–	–	–
32	**742.075**	919.704	–	–	–	–	–
64	744.563	–	–	–	–	–	–

4.2. Runtime environment evaluation

One of the key factors that influence scalability considerations is the performance of the runtime environment of the AMOS Blue Gene/Q system. The AMOS system supports both standard GNU (4.7.2) as well as `LLVM/bgclang`[5] environments [19]. Fig. 1 demonstrates the speedup for the baseline Elemental least-squares solver, built with `bgclang`, over the baseline solver built with GNU for various matrices from Table 2 on 512 BG/Q nodes. The baseline Elemental solver achieves a significant speedup in the `LLVM/bgclang` environment due to the highly optimized linear algebraic routines implemented in the BG/Q Math libraries, the ESSL (Engineering Scientific Subroutine Library), and the MASS (Mathematical Acceleration Subsystem).

To solve the least-squares regression problem, the baseline Elemental solver computes the QR decomposition of the input matrix. The QR decomposition is computed via a sequence of matrix multiplications of orthogonal matrices and the input matrix. In this setting, ESSL outperforms the BLAS routines built with the standard GNU compiler for the Blue Gene/Q system. The speedup for the Yoshiyasu matrix is much better than the one achieved for the ESOC Springer matrix; this is due to the fact that the QR decomposition depends quadratically on n and linearly on m.

4.3. Evaluating the four randomized transforms

One of the main objectives of this paper is to evaluate the Blendenpik algorithm with respect to the evaluation metrics given in Eqs. (2)–(4). We aim to understand the performance of the four

[5] http://trac.alcf.anl.gov/projects/llvm-bgq.

Please cite this article in press as: C. Iyer, et al., A randomized least squares solver for terabyte-sized dense overdetermined systems, J. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.jocs.2016.09.007

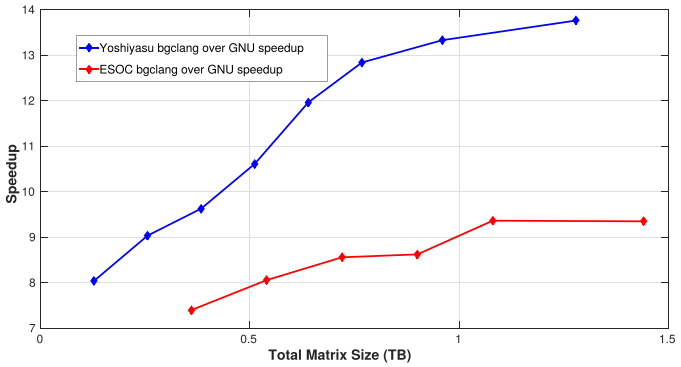


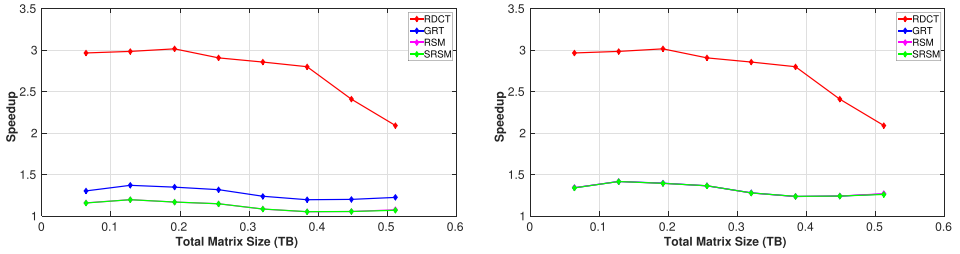
Fig. 1. Speedup defined as $t^*_{\text{gnu}}/t^*_{\text{bgclang}}$ of the baseline solver as a function of the matrix size in TBs on 512 nodes; t^*_{gnu} is the time required to solve the least-squares problem using the baseline solver and the standard GNU environment and t^*_{bgclang} is the time required to solve the least-squares problem using the baseline solver and the `LLVM/bgclang` environment.

randomized transforms $\boldsymbol{F}$ described in Section 2.1 on the Blendenpik algorithm. We also aim to understand the impact of our choice for $\boldsymbol{F}$ in terms of strong scaling and weak scaling on Blendenpik. Finally, we evaluate the impact of the oversampling factor γ on the Blendenpik algorithm using the aforementioned metrics. We set $\gamma = 2$ for evaluating our randomized transforms and we validate this choice of γ later in Section 4.4.

4.3.1. Scalability evaluation

To demonstrate that the Blendenpik algorithm is a scalable solver for tera-scale overdetermined least-squares systems, we analyze the speedup given by Eq. (4) for the Gaussian Random Transform (GRT), the Random Sign Matrix transform (RSM), the Sparse Random Sign Matrix transform (SRSM), and the Randomized Discrete Cosine Transform (RDCT) over the baseline Elemental solver. We show the scalability for each of the transforms for increasing sizes of the Yoshiyasu Mesh and the ESOC Springer dense matrices for 128 BG/Q nodes on AMOS (see Table 2). Fig. 2a and b shows the speedup achieved by the Blendenpik algorithm for the four choices of the randomized transforms for increasing sizes of the dense Yoshiyasu Mesh matrix on 128 nodes. As seen in the plots, the speedup of the RDCT easily overwhelms the other transforms for increasing matrix sizes. Another important observation is that all the randomized transforms spend a non-negligible amount of time in creating the sketching matrix $\boldsymbol{SF}$ (see Algorithm 1). Fig. 2a shows the speedup of the transforms *including* the time needed to construct the sketching matrix. The GRT performs marginally better than RSM and SRSM, as the sketching creation time for the latter transforms is higher than that of the GRT. The GRT sketching transform is implemented in BGQ using Boost's random normal distribution API as is, while the RSM and the SRSM are implemented via a lazy initialization procedure, wherein each element of the sketch is generated by a function call to Boost's discrete distribution API. This lazy initialization using function calls is responsible for the creation time overhead that is roughly similar for both the RSM and the SRSM. As seen in Fig. 2b, discounting the sketch creation time only marginally improves the speedup, while RDCT still outperforms the GRT, RSM and SRSM transforms. In all subsequent evaluations, we will include the time required to created the sketching matrix for the RDCT, since it does not significantly affect the overall running time of the resulting Blendenpik algorithm.

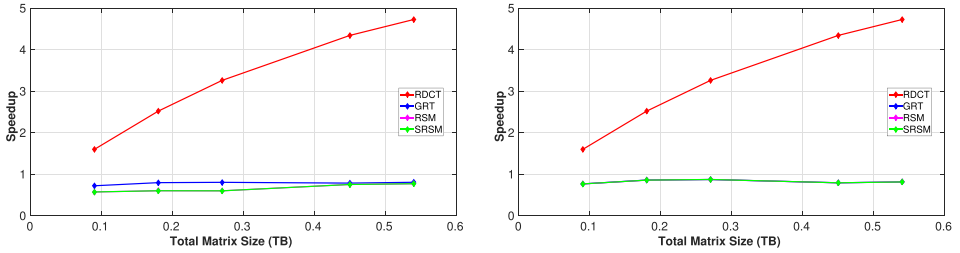
Fig. 3a and b shows the speedup achieved by the four randomized transforms (including and not including the time required to create the sketching matrix) for increasing sizes of the ESOC Springer matrix on 128 BG/Q nodes. In all cases, as the size of the ESOC matrix increases, the running times and therefore the speedup of the randomized transforms are actually worse than the baseline least squares solver; a notable and important exception is the RDCT, which outperforms the baseline solver. Another important observation is that the baseline Elemental least squares solver scales quite well for (replications of) the Yoshiyasu matrix that are highly overdetermined. Hence, the speedup observed for the Blendenpik solver using the RDCT is not as pronounced as in the previous evaluation. We did observe that for matrices that are less overdetermined (like, for example, the ESOC matrices), the speedup of RDCT improves considerably. Also the sketch creation time for the RSM and the SRSM is insignificant, especially compared to the GRT, as observed in Fig. 3a. The sketching transforms perform



(a) Speedup including the time to create the sketching matrix. Note that the RSM and the SRSM lines are overlapping.

(b) Speedup *not* including the time to create the sketching matrix (except for the RDCT). Note that the GRT, RSM, and SRSM lines are overlapping.

Fig. 2. Speedup for dense matrices as a function of increasing matrix size for the Yoshiyasu Mesh matrix on 128 BG/Q nodes.

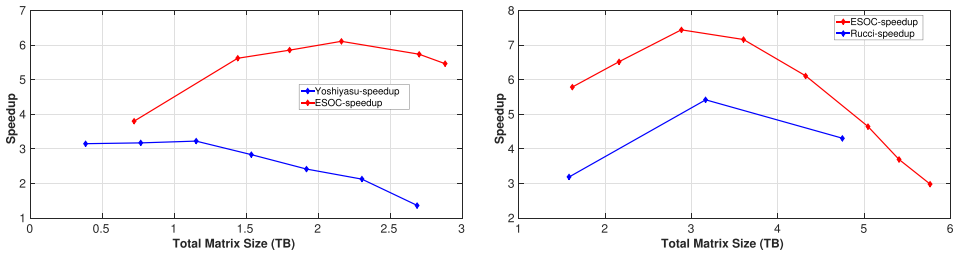


(a) Speedup including the time to create the sketching matrix. Note that the RSM and the SRSM lines are overlapping.

(b) Speedup *not* including the time to create the sketching matrix (except for the RDCT). Note that the GRT, RSM, and SRSM lines are overlapping.

Fig. 3. Speedup analysis for dense matrices as a function of increasing matrix size for the ESOC Springer matrix on 128 BG/Q nodes.

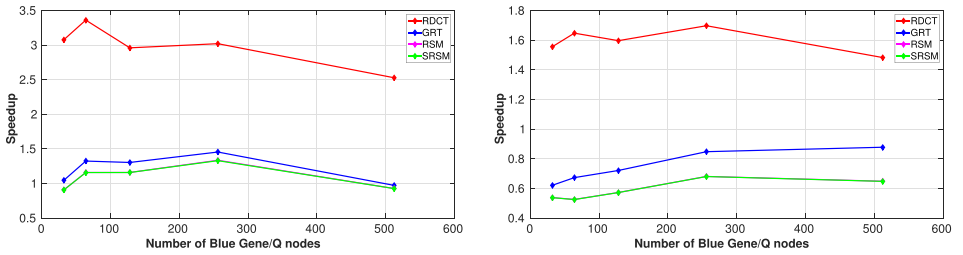
Please cite this article in press as: C. Iyer, et al., A randomized least squares solver for terabyte-sized dense overdetermined systems, J. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.jocs.2016.09.007



(a) Speedup for the Yoshiyasu Mesh and ESOC Springer matrices on 512 BG/Q nodes.

(b) Speedup for the ESOC Springer and Rucci matrices on 1,024 BG/Q nodes.

Fig. 4. Speedup for dense matrices as a function of increasing matrix size.



(a) Strong scaling for the Yoshiyasu Mesh-4 matrix $(936,092 \times 9,393)$. The RSM and SRSM lines overlap.

(b) Strong scaling for the ESOC Springer matrix $(327,062 \times 37,830)$. The RSM and SRSM lines overlap.

Fig. 5. Strong scaling as a function of the number of Blue Gene/Q nodes used in our evaluation.

comparably when the time to construct the sketching matrix is not included (see Fig. 3b).

4.3.2. Scalability evaluation for terascale matrices

The key objective of our work here is to evaluate the Blendenpik algorithm as a solver for terascale overdetermined least-squares problems. As seen in Section 4.3.1, using the RDCT transform in the Blendenpik algorithm outperforms the other randomized transforms even for moderately sized matrices. Hence, we only evaluate the scalability of Blendenpik using the RDCT for terascale matrices. Fig. 4a and b shows the scalability of our solver on 512 nodes and 1024 nodes respectively on AMOS. As discussed earlier, the base Elemental least-squares solver scales quite well for highly overdetermined dense matrices. Hence the speedup observed for the Blendenpik solver compared to the baseline solver is not as significant as seen in Fig. 4a. However, for matrices that are less overdetermined, the runtime and thus the speedup of the Blendenpik algorithm improves considerably. This observation is obvious for the ESOC Springer matrix, shown in Fig. 4b.

One observation that is particularly significant is the effect of batchwise RDCT on the speedup as the matrix size increases. The number of columns transformed in a single batch depends on the number of rows of the matrix as well as on the minimum space available across all processes to allocate the columns. Thus, as the number of rows increases, fewer and fewer columns fit in a batch making the batchwise transformation step slower. This is the reason underlying the observation that the speedup peaks at the point where the entire transformed matrix is able to fit into memory and beyond this stage, the batchwise processing slowdown kicks in. This effect is more pronounced for an increasing number of replications of the ESOC Springer matrix and the Rucci matrix in Fig. 4b on 1024 BG/Q nodes.

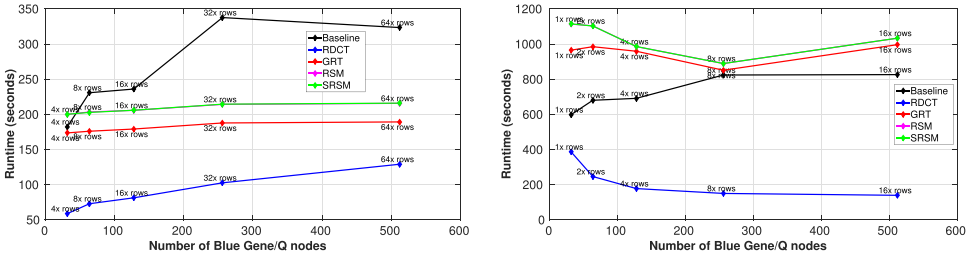
In general, the Blendenpik algorithm scales excellently as compared to the baseline Elemental solver. While the baseline solver fails to execute for the Yoshiyasu-192 (see Table 2) on 512 nodes, as well as for the ESOC-36 (ESOC matrix with 36 replications) in 512 nodes and the ESOC-68 (ESOC matrix with 68 replications) in 1024 nodes, the Blendenpik solver is able to scale to such matrix sizes. Another important aspect of the Blendenpik algorithm for the terascale matrices from Table 2 is that the steps 11–12 is never performed as the densification step of constructing these matrices generates moderately well-conditioned matrices, and hence the preconditioner constructed is also well-conditioned.

4.3.3. Performance evaluation

We also evaluate the strong and weak scaling performance of the Blendenpik algorithm as a function of the (increasing) number of the Blue Gene/Q nodes. Fig. 5a and b shows the strong scaling performance of the four randomized sketching transforms on the Yoshiyasu-4 matrix (i.e., the base Yoshiyasu Mesh matrix replicated four times), as well as on the base ESOC Springer dense matrix, respectively. We observe that the speedup of the Blendenpik solver increases marginally with an increasing number of BG/Q nodes; this effect is more pronounced in the case of the Yoshiyasu Mesh matrix rather than the ESOC Springer matrix. However, this advantage is offset as the baseline Elemental solver performs comparably to the batchwise Blendenpik solver for 512 BG/Q nodes. This slowdown in the batchwise Blendenpik solver is mainly because of the QR preconditioning phase that does not scale as well as the randomized sketching matrices and the LSQR stages of the Blendenpik algorithm, as the number of BG/Q nodes increases. The RDCT outperforms the other sketching transforms by at least a factor of two for all BG/Q node configurations.

Finally, the weak scaling performance of the Blendenpik algorithm on the Yoshiyasu Mesh and ESOC Springer matrices for the four randomized sketching transforms that we evaluate in this work is shown in Fig. 6a and b, respectively. We observe that the runtime for the RDCT on the Yoshiyasu Mesh matrix increases sublinearly as the matrix size increases and as the number of BG/Q nodes increases; at the same time, there is a significant bump in the running time of the baseline solver. Also, the runtimes for the other randomized sketching transforms (GRT, RSM, and SRSM) remain approximately constant as the matrix size and the number of BG/Q nodes both increase. Furthermore, the runtimes for the randomized sketching transforms are much better than the runtime of the baseline solver. Interestingly, the runtime of the RDCT for the ESOC matrix keeps diminishing, even as the number of rows and the number of BG/Q nodes keeps increasing. As we already discussed in

Please cite this article in press as: C. Iyer, et al., A randomized least squares solver for terabyte-sized dense overdetermined systems, J. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.jocs.2016.09.007



(a) Weak scaling for the Yoshiyasu Mesh matrix $(234,023 \times 9,393)$. (b) Weak scaling for the ESOC Springer matrix $(327,062 \times 37,830)$.

Fig. 6. Weak scaling as a function of matrix sizes and number of Blue Gene/Q nodes.

the strong scaling analysis, the primary bottleneck for the reduction in performance as the number of BG/Q nodes increases has to do with the runtime of the QR decomposition at the preconditioning stage. However, the size of the sampled matrix, which is the input to the QR decomposition at the preconditioning stage, is independent of the number of rows of the original input matrix. As additional BG/Q nodes are allocated, the performance of the QR decomposition at the preconditioning stage improves. This boosts the overall runtime of the Blendenpik algorithm, an effect that is observed for all four randomized sketching transforms, even though it is much more pronounced for the RDCT.

4.3.4. Numerical stability evaluation

We evaluate the numerical stability of the Blendenpik solver for all randomized sketching transforms as the matrix size increases. The numerical stability is captured by the relative error (see Eq. (2)) and the backward error (see Eq. (3)). Our evaluations on 128 BG/Q nodes show a relative error within 11-12 digits of accuracy for all randomized sketching transforms for both the ESOC and Yoshiyasu matrices. Thus, these values are much better than $O(\sqrt{\epsilon_{\text{machine}}})$, which is well within the bounds on the relative error guarantees given by Drineas et al. [6]. We skip the relative error plots in the interest of space and describe the numerical stability of the Blendenpik solver captured by the backward error instead. Fig. 7a and b captures the behavior of the backward error as the size of the Yoshiyasu Mesh and ESOC Springer matrices increases. We observe that the backward error for all randomized transforms is roughly two orders of magnitude worse than the baseline solver. Furthermore, the backward error for all transforms, including the baseline solver for the ill-conditioned ESOC Springer matrix, is several orders of magnitude worse (approximately five orders of magnitude worse) than the backward error for the relatively well-conditioned Yoshiyasu Mesh matrix. While the relative error captures the stability of the solution, the backward error captures the stability of the system, and the more ill-conditioned the system is, the worse the error will be.

4.3.5. Numerical stability evaluation for terascale matrices

We evaluate the numerical stability for the Blendenpik solver using the relative error metric of Eq. (2) for increasing matrix sizes and for 512 Blue Gene/Q nodes, for both the Yoshiyasu Mesh and the ESOC matrices; see Fig. 8a. We only show results for the RDCT, since all four randomized sketching transforms have approximately the same behavior for both the relative and the backward error. We observe that the relative error is again well within the $O(\sqrt{\epsilon_{\text{machine}}})$ bounds. The numerical stability defined by backward error is shown in Fig. 8b for both the baseline Elemental solver and the Blendenpik solver. We observe that the ESOC Springer matrix has worse backward error than the Yoshiyasu Mesh matrix (approximately five orders of magnitude worse) for increasing matrix sizes; this is due to its high condition number. However, the backward error of the Blendenpik solver is comparable to the backward error of the baseline solver. This error could potentially be improved by either using more than one preprocessing stages or by selecting larger sample sizes for the preconditioning stage. The latter choice would lead to worse running times and reduced speedups as the size of the input matrices increases. Another approach to overcome this tradeoff is to apply a random sketching transform matrix $\boldsymbol{F}$ as proposed in the ground-breaking paper of Clarkson and Woodruff [20] and then apply the RDCT to $\boldsymbol{FA}$. We refer the reader to [20] for a detailed description of their original construction and simply note that applying the resulting matrix $\boldsymbol{SF}$ on the input matrix $\boldsymbol{A}$ takes time proportional to the sparsity of the input matrix $\boldsymbol{A}$.

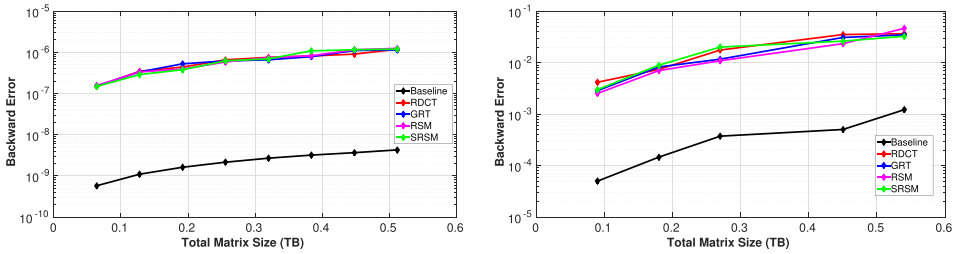
4.4. The effect of the oversampling factor γ

An important choice in the construction of an efficient preconditioner in the context of the Blendenpik algorithm is 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 various randomized sketching transforms in the Blendenpik solver with respect to the metrics described in Section 4 as a function of γ. We evaluate the Blendenpik solver on the Yoshiyasu-12 and the ESOC-4 matrices on 128 BG/Q nodes as a function of γ, where γ ranges between 1.5 and six in increments of 0.5. We seek to understand the effect of γ on the scalability and the numerical stability of the Blendenpik algorithm.

Fig. 9a and b shows the speedup of the Blendenpik algorithm for increasing values of the oversampling factor γ for the various randomized sketching transforms on the Yoshiyasu-12 and the ESOC-4 matrices, respectively. Fig. 9a reveals several interesting observations as the oversampling factor γ increases. The speedup of the RDCT increases marginally as the value of γ increases. This is because the computational cost of applying the RDCT dominates the QR preconditioning and the LSQR stages for highly overdetermined matrices. As the oversampling factor increases, the transformation time remains the same, while the computational time of the QR decomposition, which is comparatively much smaller, increases. Furthermore, as the oversampling factor γ increases, a better preconditioner is constructed, which leads to a faster convergence time for LSQR. However, the speedups of the other randomized sketching transforms decrease, mainly due to the dominant cost of the time that it takes to apply the random sketching transformation as the oversampling factor γ increases.

Fig. 9b shows the monotonically decreasing speedup of the RDCT as the oversampling factor γ increases for the ESOC-3 matrix. This is due to the computational cost of the QR preconditioning stage, which dominates the computational time needed to apply the randomized sketching matrix as well as the LSQR solver stage, since the input matrix is not as overdetermined as the Yoshiyasu-12 matrix. As the oversampling factor γ increases, the time to compute the QR decomposition in the preconditioning stage also increases, leading to an overall reduced speedup. This behavior is also exhibited by the other randomized sketching transforms. Furthermore, the speedup of the RDCT easily overwhelms the speedup of the other

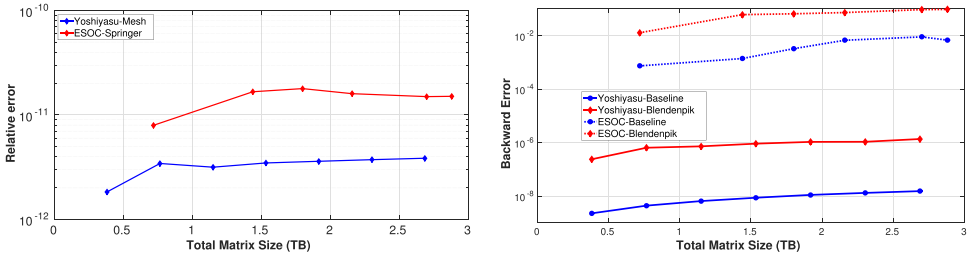
Please cite this article in press as: C. Iyer, et al., A randomized least squares solver for terabyte-sized dense overdetermined systems, J. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.jocs.2016.09.007



(a) Numerical stability for different randomized sketching transforms for increasing sizes of the Yoshiyasu Mesh matrix.

(b) Numerical stability for different sketching transforms for increasing sizes of the ESOC Springer matrix .

Fig. 7. Numerical stability (backward error analysis) as a function of (increasing) matrix sizes for the Yoshiyasu Mesh and the ESOC Springer matrices for 128 BG/Q nodes.



(a) Numerical stability: relative error.

(b) Numerical stability: backward error.

Fig. 8. Numerical stability as a function of matrix size for the Yoshiyasu Mesh and the ESOC Springer matrices on 512 BG/Q nodes.

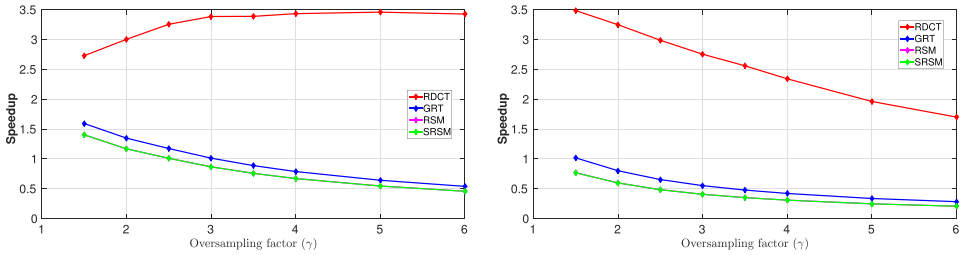
randomized sketching transforms for increasing values of the oversampling factor γ.

As discussed in Section 4.3.4, the numerical stability is measured in terms of relative and backward error. Also, again as discussed earlier, the relative error for all randomized sketching transforms for both the ESOC and the Yoshiyasu matrices is within 11-12 digits of accuracy, and hence we measure the numerical stability in terms of the backward error only.

Fig. 10a and b shows the behavior of the backward error as a function of the oversampling factor γ for the Yoshiyasu-12 and the ESOC-3 matrix, respectively. It is worth noting that the backward error for both matrices sharply decreases for all randomized sketching transforms at $\gamma = 2$ and monotonically continues to decrease as the oversampling factor γ increases. Thus, γ equal to two acts as a knee point validating our choice for γ for our tera-scale evaluations. All the randomized sketching transforms exhibit errors that are approximately within the same order of magnitude for the various choices of the oversampling factor γ.

4.5. Summarizing our empirical evaluations

To help the reader parse our extensive empirical evaluations, we briefly summarize our findings. (i) The Randomized Discrete Cosine Transform (RDCT) outperforms the Gaussian Random Transform (GRT), the Random Sign Matrix Transform (RSM) and the Sparse Random Sign Matrix Transform (SRSM) in terms of scalability and performance. (ii) The computational cost of the various stages of the Blendenpik solver is determined by how overdetermined the input matrix is. The more overdetermined the matrix, the higher the computational cost of the random sketching transform stage. As the matrix becomes less overdetermined, the running time of the QR decomposition in the preconditioning stage becomes more and more dominant. This is especially true for the RDCT. (iii) The scalability of the batchwise Blendenpik implementation is determined by the number of columns in each batch of the RDCT transform, which in turn is determined by the number of rows of the matrix. As the number of rows increases, the runtime of the batchwise sketching transform stage worsens, leading to reduced speedups. (iv) The batchwise Blendenpik solver using the RDCT demonstrates significant strong and weak scaling for all matrices. (v) The Blendenpik solver demonstrates excellent numerical stability in terms of the forward error for increasing matrix sizes. The backward error is somewhat worse yet comparable to the backward error achieved by the baseline Elemental solver. (vi) The oversampling factor γ determines the quality of the preconditioner for the Blendenpik solver. Choosing a higher oversampling factor leads to better numerical stability. However, higher oversampling factors lead to a reduced performance. This tradeoff becomes less significant as the input matrix becomes more overdetermined.

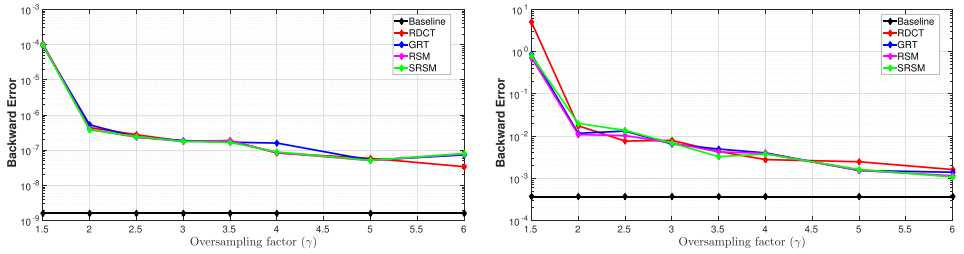


(a) Speedup for the Yoshiyasu-12 matrix. The RSM and SRSM lines overlap.

(b) Speedup for the ESOC-3 matrix. The RSM and SRSM lines overlap.

Fig. 9. Speedup as a function of the (increasing) oversampling factor γ for the Yoshiyasu-12 matrix and the ESOC-3 matrix on 128 BG/Q nodes.

Please cite this article in press as: C. Iyer, et al., A randomized least squares solver for terabyte-sized dense overdetermined systems, J. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.jocs.2016.09.007



(a) Backward error for different randomized sketching transforms for the Yoshiyasu-12 matrix ($2,808,276 \times 9,393$).

(b) Backward error for different randomized sketching transforms for the ESOC-3 matrix ($981,186 \times 37,830$).

Fig. 10. Accuracy analysis in terms of backward error as a function of increasing oversampling factors for the Yoshiyasu-12 matrix and the ESOC-3 matrix on 128 BG/Q nodes.

5. Conclusions and future work

We implemented and thoroughly evaluated a highly scalable, distributed memory, least-squares solver based on the Blendenpik algorithm. Our solver, which is based on an implementation of the Blendenpik algorithm in a distributed setting coupled with various batchwise transformations in order to construct an appropriate preconditioner, beats state-of-the-art least-squares solvers with respect to running time and scales to much larger matrices compared to prior work. In future work, we plan to explore the possibility of reducing the communication overhead involved in the randomized transformations that are used in the preconditioner construction.

References

[1] K.A. Gallivan, R.J. Plemmons, A.H. Sameh, Parallel algorithms for dense linear algebra computations, SIAM Rev. 32 (1990) 54–135.
[2] J. Demmel, L. Grigori, M. Hoemmen, J. Langou, Communication-optimal parallel and sequential QR and LU factorizations. UC Berkeley Technical Report EECS- 2008-89, Aug 1, 2008, Submitted to SIAM. J. Sci. Comp., 2008.
[3] J. Demmel, K. Yelick, Communication Avoiding (CA) and Other Innovative Algorithms, The Berkeley Par Lab: Progress in the Parallel Computing Landscape, 2014, pp. 243–250.
[4] P. Drineas, M.W. Mahoney, RandNLA: Randomized numerical linear algebra, Commun. ACM 59 (2016) 80–90.
[5] T. Sarlos, Improved approximation algorithms for large matrices via random projections, in: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, FOCS'06, IEEE Computer Society, Washington, DC, USA, 2006, pp. 143–152.
[6] P. Drineas, M.W. Mahoney, S. Muthukrishnan, T. Sarlós, Faster least squares approximation, Numer. Math. 117 (2011) 219–249.
[7] V. Rokhlin, M. Tygert, A fast randomized algorithm for overdetermined linear least-squares regression, Proc. Natl. Acad. Sci. U.S.A. 105 (2008) 13212–13217.
[8] H. Avron, P. Maymounkov, S. Toledo, Blendenpik: Supercharging LAPACK's least-squares solver, SIAM J. Sci. Comput. 32 (2010) 1217–1236.
[9] J. Yang, X. Meng, M.W. Mahoney, Implementing Randomized Matrix Algorithms in Parallel and Distributed Environments, CoRR abs/1502.03032, 2015.
[10] X. Meng, M.A. Saunders, M.W. Mahoney, LSRN: A parallel iterative solver for strongly over- or under-determined systems, CoRR abs/1109.5981, 2011.
[10] M.P. Forum, MPI: A Message-Passing Interface Standard, Technical Report, Knoxville, TN, USA, 1994.
[12] L. Dagum, R. Menon, OpenMP: an industry-standard API for shared-memory programming, IEEE Comput. Sci. Eng. 5 (1998) 46–55.
[13] C.C. Paige, M.A. Saunders, LSQR: an algorithm for sparse linear equations and sparse least squares, ACM Trans. Math. Softw. 8 (1982) 43–71.
[14] D. Achlioptas, Database-friendly random projections: Johnson-Lindenstrauss with binary coins, J. Comput. Syst. Sci. 66 (2003) 671–687.
[15] J. Poulson, B. Marker, R.A. van de Geijn, J.R. Hammond, N.A. Romero, Elemental: a new framework for distributed memory dense matrix computations, ACM Trans. Math. Softw. 39 (2013), 13:1–13:24.
[16] M. Frigo, S.G. Johnson, FFTW: an adaptive software architecture for the FFT, in: Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, volume 3, Seattle, Washington, 1998, pp. 1381–1384.
[17] J.R. Hammond, A. Schäfer, R. Latham, To INT_MAX.. and beyond!: Exploring large-count support in MPI, in: Proceedings of the 2014 Workshop on Exascale MPI, ExaMPI'14, IEEE Press, Piscataway, NJ, USA, 2014, pp. 1–8.
[18] T.A. Davis, Y. Hu, The University of Florida sparse matrix collection, ACM Trans. Math. Softw. 38 (2011), 1:1–1:25.
[19] C. Lattner, V. Adve, LLVM: a compilation framework for lifelong program analysis & transformation, in: Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04), Palo Alto, California, 2004.
[20] K.L. Clarkson, 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, STOC'13, ACM, New York, NY, USA, 2013, pp. 81–90.

Chander Iyer is currently a 4th year Ph.D. student in the Department of Computer Science at Rensselaer Poly-technic Institute. He received his B.E. from Mumbai University, Mumbai, India, in 2003 and received his M.Tech. in 2010 from Indian Institute of Technology, Bombay. He is currently being advised by Prof. Petros Drineas, with Prof. Chris Carothers as his co-advisor. His research interests lie at the intersection of Randomized Algorithms for large scale datasets, High Performance Computing and Machine Learning.

Haim Avron did his Ph.D. at the School of Computer Science at Tel Aviv University under the supervision of Prof. Sivan Toledo. Afterwards he spent two years as a Postdoctoral Researcher in the Business Analytics & Mathematical Sciences department at the IBM T.J. Watson Research Center. From 2012 to 2015 he was a Research Sta Member in the Mathematical Sciences & Analytics department at the IBM T.J. Watson Research Center. He joined the Department of Applied Mathematics, School of Mathematical Sciences at Tel Aviv University as a Senior Lecturer (equivalent to assistant professor) in 2015. His research focuses on numerical computing and high performance computing and their applications in scientific computing and machine learning. His interests and work range from mathematical and computational foundations to end-to-end implementation aspects.

Georgios Kollias received the B.Sc. in Physics in 2000 and the M.Sc. in Computational Science in 2002 from the University of Athens, Greece, and the PhD in Computer Science from the University of Patras, Greece, in 2009. He moved to Purdue University, USA in October 2009 and worked as a Postdoctoral Researcher in the Computer Science Department and the Center for Science of Information till May 2013. Then he joined IBM T.J. Watson Research Center, USA and in August 2014 he moved to IBM Zurich Research Lab. He returned back in IBM T.J. Watson Research Center in April 2015 as a Research Sta Member in the area of Big Data Management and Analytics. His research interests include Parallel, Distributed and High Performance Computing, Numerical Linear Algebra and Matrix Computations, Graph Mining, Data Analytics and Problem Solving Environments.

Yves Ineichen received his M.Sc. in Computer Science in 2008, and the Phd in Computer Science in 2013 from the Federal Institute of Technology Zurich (ETHZ), Switzerland. In the beginning of 2013 he joined the IBM Research Center in Rueschlikon, Switzerland. His research interests include: High Performance Computing, Optimization, Numerical Linear Algebra, Compiler Design, Programming Languages. Yves is a recipient of the PRACE (2012) and ACM Gordon Bell (2015) award.

Please cite this article in press as: C. Iyer, et al., A randomized least squares solver for terabyte-sized dense overdetermined systems, J. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.jocs.2016.09.007

Professor Chris Carothers is a faculty member in the Computer Science Department at Rensselaer Polytechnic Institute. He received the Ph.D., M.S., and B.S. from Georgia Institute of Technology in 1997, 1996, and 1991, respectively. Prior to joining RPI in 1998, he was a research scientist at the Georgia Institute of Technology. His research interests are focused on massively parallel computing which involve the creation of high fidelity models of extreme-scale networks and computer systems. These models have executed using nearly 2,000,000 processing cores on the largest leadership class supercomputers in the world. Professor Carothers is an NSF CAREER Award winner as well as Best Paper award winner at the ACM-SIGSIM PADS Conference for 1999, 2003 and 2009. Since joining Rensselaer, he has developed a world-class research portfolio which includes funding from the NSF, the U.S. Department of Energy, Army Research Laboratory, Air Force Research Laboratory, as well as several companies, including IBM, General Electric, and AT&T.

Additionally, Professor Carothers serves as the Director for the Rensselaer Center for Computational Innovations (CCI). CCI is a partnership between Rensselaer and IBM. The center provides computation and storage resources to diverse network of researchers, faculty, and students from Renssleaer, government laboratories, and companies across a number of science and engineering disciplines. The agship supercomputer is a 1 petaFLOP IBM Blue Gene/Q system with 80 terabytes of memory, 81,920 processing cores and over 2 petabytes of disk storage.

Professor Petros Drineas is an Associate Professor at the Computer Science Department of Rensselaer Polytechnic Institute. Prof. Drineas earned a PhD in Computer Science from Yale University in May of 2003, and a BS in Computer Engineering and Informatics from the University of Patras, Greece, in July of 1997. Prof. Drineas' research interests lie in the design and analysis of randomized algorithms for linear algebraic problems, as well as their applications to the analysis of modern, massive datasets. Prof. Drineas received an NSF CAREER in 2006; was a Visiting Professor at the US Sandia National Laboratories during the fall of 2005; was a Visiting Fellow at the Institute for Pure and Applied Mathematics at the University of California, Los Angeles in the fall of 2007; and was a Visiting Professor at the University of California Berkeley in the fall of 2013. Prof. Drineas has also served the US National Science Foundation (NSF) as a Program Director in the Information and Intelligent Systems (IIS) Division and the Computing and Communication Foundations (CCF) Division (2010–2011). Prof. Drineas has published over 90 articles in conferences and journals in Theoretical Computer Science, Numerical Linear Algebra, and statistical data analysis.

Please cite this article in press as: C. Iyer, et al., A randomized least squares solver for terabyte-sized dense overdetermined systems, J. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.jocs.2016.09.007