Solving Large Scale Systems Of Linear Equations with A Stabilized Lanczos-Type Algorithms Running On A Cloud Computing Platform

Maharani, M.[*] and Abdellah Salhi[†] and Wali Khan Mashwani[‡ §] and Ozgur Yeniay[¶] and Niken Larasati [‖] and Triyani [**]

Abstract

Lanczos-type algorithms for Systems of Linear Equations (SLEs) are efficient but fragile. A number of ways to resolve this issue have been suggested. But, the problem is still not fully sorted, in our view. Here, we suggest a way that takes advantage of the sequence of approximate solutions that have been computed prior to breakdown by embedding interpolation/extrapolation to avoid it. The approach, referred to as Embedded Interpolation-Extrapolation Model in Lanczos-type Algorithm (EIEMLA), generates new iterates which are at least as good as the best in the current sequence. This process is repeated after appending the new iterates to the sequence of approximate solutions until some convergence tolerance is achieved. To improve EIEMLA's convergence and stability, a restart version of REIEMLA is also considered. These algorithms are more robust than other Lanczos-type algorithms, including those with restarting and switching strategies. Both algorithms have been implemented to run in parallel on a Cloud computing platform. Our tests involve SLEs with up to 10^6 variables and equations. The results show that breakdown is mitigated and efficiency gains can be achieved through parallelization.

Keywords: Lanczos-type algorithm, Breakdown, EIEMLA method, Parallel processing, Cloud computing.

2000 AMS Classification: AMS

[*]Department of Mathematics, University of Jenderal Soedirman, Purwokerto, Indonesia, Email: maharani@unsoed.ac.id

[†]Department of Mathematical Sciences, University of Essex, Colchester, CO4 3SQ, U.K., Email: as@essex.ac.uk

[‡]Department of Mathematics, Kohat University of Science and Technology, Pakistan, Email: mashwanigr8@gmail.com

[§]Corresponding Author.

[¶]Department of Statistics, Hacettepe University Ankara, Turkey, Email: yeniay@hacettepe.edu.tr

[‖]Department of Mathematics, University of Jenderal Soedirman, Purwokerto, Indonesia, Email: nklarasati@yahoo.com

[**]Department of Mathematics, University of Jenderal Soedirman, Purwokerto, Indonesia, Email: trianisr@yahoo.com.au

1. INTRODUCTION

Large scale Systems of Linear Equations (SLEs) occur routinely in a variety of applications ranging from engineering to finance and economics. It is therefore, important to investigate methods which handle such problems. Lanczos method is one of effective iterative methods which deals with high dimension of SLEs, [22, 23]. The original derivation, however, still used classical algebra. The modern version of Lanczos method, well-known as Lanczos-type algorithms, were derived through theory of formal orthogonal polynomials and realized via recurrence relationships between them, [3, 5]. Although Lanczos-types algorithms are effective to solve non-symmetric and high scale of SLEs, however, they are easily to breakdown which makes the algorithms stop before reaching a good solution. It is therefore, a strategy to get over the breakdown so that Lanczos-types algorithms maintain their stability to convergent, is importantly needed.

Some approaches to cure breakdown in Lanczos-types algorithms have been developed. For instance, Brezinski and his team have established a method called Method of Recursive Zoom (MRZ), which allows to jump over the non-existing orthogonal polynomial [6]. This method is also known as a look ahead strategy. Some variants of MRZ such as SMRZ, BMRZ, GMRZ, and BSMRZ have also been developed in [7] and [8]. Other techniques which also deal with this concern, are the called look around strategies, [19] and in [1]. The recent works, [15, 16], discussed the strategies of restarting and switching between Lanczos-type algorithms. Restarting which based on qualitative points, including a point from the last iterate, a point with the lowest residual norm, and a point with median value, was investigated in [25]. Here, we introduce a new approach to improve the resilience and stability of Lanczos-types algorithms for solving SLEs, by capturing some existing patterns in the sequences of solutions generated by Lanczos-type algorithms. This new approach is hence called the embedded interpolation and extrapolation model in Lanczos-type algorithm (EIEMLA).

On the other hand, the solution of large scale SLEs is time consuming, particularly on single processor machines. Therefore, we also will try to exploit multiple processors and powerful vector-oriented hardware to tackle this issue. In general, parallel algorithms can be created by reformulating standard algorithms or by discovering new ones, [20]. The implementation of parallel iterative methods for solving SLE's in high dimensions and other applications is well developed, [11, 10, 27, 28, 32, 30]. Here, we rely on a parallel environment provided in Matlab to implement our algorithms and run them. Moreover, we will execute the parallel program of EIEMLA on the cloud computing.

This study is organized as follow. Section 1 discusses the back ground review of Lanczos-type algorithms, Section 2 introduces the derivation of EIEMLA. Sections 3 and 4 explain how to implement and run EIEMLA in a parallel environment and on a Cloud computing platform, respectively. Section 5 discusses some numerical results, concludes this work and suggests interesting and related avenues for further research.

2. Lanczos-type Algorithms : Review

Lanczos-type are effective iterative method for solving non symmetric and high dimension of systems of linear equations (SLEs). However, they are fragile when involving high number of iterations. Here, we review the derivation of Lanczos-type methods through Krylov subspace method, [29].

Consider the systems of linear equations

$$A\mathbf{x} = \mathbf{b}, \tag{2.1}$$

where $A \in R^{n\times n}$ and $\mathbf{x}, \mathbf{b} \in \mathbf{R}^n$. Krylov subspace method finds the approximate solutions by defining (1) $\mathbf{x}_k - \mathbf{x}_0 \in K_k(A, \mathbf{r}_0)$ which the residual vector, (2) $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$, is orthogonal to Krylov subspace $K_k(A^T, \mathbf{y})$, where $\mathbf{y}$ is an arbitrary non-zero vector. As a result of (1) and (2), we have :

$$\mathbf{x}_k - \mathbf{x}_0 = -\alpha_1\mathbf{r}_0 - \alpha_1 A\mathbf{r}_0 - \cdots - \alpha_k A^{k-1}\mathbf{r}_0. \tag{2.2}$$

and

$$\mathbf{r}_k = \mathbf{r}_0 + \alpha_1 A\mathbf{r}_0 + \alpha_2 A^2\mathbf{r}_0 + \cdots + \alpha_k A^k\mathbf{r}_0 \tag{2.3}$$

If $P_k(A) = 1 + \alpha_1 A + \cdots + \alpha_k A^k$, of degree k at most, then $\mathbf{r}_k$ can be expressed as :

$$\mathbf{r}_k = P_k(A)\mathbf{r}_0. \tag{2.4}$$

The P_k is hence called Orthogonal polynomial which is normalized by $P_k(0) = 1$. To simplify the computation of coefficients of P_k, we define a linear function c, which satisfy two conditions, [4],

$$c_i = c(t^i), \quad i = 0, 1, \ldots, \tag{2.5}$$

and

$$c(t^i P_k(t)) = 0, \quad i = 0, 1, \ldots, k-1. \tag{2.6}$$

By this definition, we can set $c_i = \langle \mathbf{y}, A^i\mathbf{r}_0\rangle$ and thus the coefficients of $P_k(A)$ can be computed recursively. Thus, the approximate solutions are determined by formula $\mathbf{x}_k = \mathbf{b} - A\mathbf{r}_k$, where $\mathbf{r}_k = P_k(A)\mathbf{r}_0$, without inverting the matrix A. Some Lanczos-type algorithms are then implemented based on some formula derived by those approaches.

Those derivations allow us to explore more and more formula which Lanczos-type algorithms based. Some of them have been discussed in [2],[3], and [9], which are expressed in a table containing formulas A_i and B_j. The new Lanczos-types implementation discovered by Farooq, [14], which involve the polynomials with the difference in degrees between left and right sides is at most two or three.

3. Embedding Interpolation and Extrapolation in Lanczos-type Algorithms (EIEMLA)

Embedding an interpolation model within a Lanczos-type algorithm is a novel strategy which produces sequences of solutions which are better than those generated by the original Lanczos algorithm.

Suppose we run a Lanczos-type algorithm , [2, 3], and stop after k iterations and before breakdown occurs. Let $k \leq n$, where n is the dimension of the SLE

in hand. Let $S = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$ be the set of iterates generated so far. Let $\mathbf{x}_m$, be the iterate with the lowest residual norm, $\|\mathbf{r}_m\|$, where $m \leq k$. Assume that some good iterates, namely those with small residual norms, are concentrated in interval $[m-j,k]$, for some integer j. Set

$$V_1 = \left\{\mathbf{x}_{m-j}, \mathbf{x}_{m-j+1}, \ldots, \mathbf{x}_k\right\}, \tag{3.1}$$

which is a subset of S. Write the components of each iterate in S as

$$\begin{aligned} v_1 &= \left\{{x_{m-j}}^{(1)}, {x_{m-j+1}}^{(1)}, \ldots, {x_k}^{(1)}\right\} \\ v_2 &= \left\{{x_{m-j}}^{(2)}, {x_{m-j+1}}^{(2)}, \ldots, {x_k}^{(2)}\right\} \\ &\vdots \\ v_n &= \left\{{x_{m-j}}^{(n)}, {x_{m-j+1}}^{(n1)}, \ldots, {x_k}^{(n)}\right\} \end{aligned} \tag{3.2}$$

namely, each v_i contains all of the i^{th} entries of iterates $\mathbf{x}_l$, for $l = m-j, m-j+1, \cdots, k$, and for $i = 1,2,\cdots,n$. It is now possible to find a function which interpolates each set of v_i's using the PCHIP interpolant package, [18, 17]. We assume that each sequence of ${x_{m-j}}^{(i)}, {x_{m-j+1}}^{(i)}, \ldots, {x_k}^{(i)}$ is monotonic and convergent for some j and $i = 1,2,\ldots,n$, to its limit, [31], i.e.

$$\lim_{k\to\infty} {x_k}^{(i)} = {x_*}^{(i)}. \tag{3.3}$$

Let t be elements in R. Set

$$\begin{aligned} w_1 &= \left\{\left(t_{m-j}, {x_{m-j}}^{(1)}\right), \left(t_{m-j+1}, {x_{m-j+1}}^{(1)}\right), \ldots, \left(t_k, {x_k}^{(1)}\right)\right\} \\ w_2 &= \left\{\left(t_{m-j}, {x_{m-j}}^{(2)}\right), \left(t_{m-j+1}, {x_{m-j+1}}^{(2)}\right), \ldots, \left(t_k, {x_k}^{(2)}\right)\right\} \\ &\vdots \\ w_n &= \left\{\left(t_{m-j}, {x_{m-j}}^{(n)}\right), \left(t_{m-j+1}, {x_{m-j+1}}^{(n)}\right), \ldots, \left(t_k, {x_k}^{(n)}\right)\right\}. \end{aligned} \tag{3.4}$$

Using PCHIP, [18, 17] to interpolate each w_i, for $i = 1,2,\ldots,n$, yields functions f_i. As it is a regular interpolation process in R, then for some $t = m-j, m-j+1, \ldots, k$, f_i satisfies

$$f_i(t) \approx x_t^{(i)} \qquad \text{for } i = 1,2,\ldots,n. \tag{3.5}$$

For instance,

$$\begin{aligned} f_i(m-j) &\approx x_{m-j}^{(i)} \\ f_i(m-j+1) &\approx x_{m-j+1}^{(i)} \\ &\vdots \\ f_i(k) &\approx x_k^{(i)} \qquad \text{for } i = 1,2,\ldots,n. \end{aligned} \tag{3.6}$$

Since we use an appropriate interpolant to interpolate the data, i.e. the one that preserves the monotonicity of the data, then the extrapolation based on this interpolation process enables us to get the next point outside of the range. It

means that if we calculate $f_i(t^*)$ with $t^* \in [k+1, s] \subset R$, where $s \geq k+1$, then we obtain

$$f_i(t^*) \approx x_r^{(i)} \qquad \text{for } i = 1, 2, \ldots, n, \tag{3.7}$$

where each $x_r^{(i)}$ has a similar property as $x_t^{(i)}$ in (3.5). In other words, if the sequence of $x_t^{(i)}$ is monotonically increasing/decreasing, so is $x_r^{(i)}$. Thus arranging vector $\mathbf{x}_r$, with $x_r{}^{(i)}$ being the i^{th} entries of the vector, yields an approximate solution of the system.

Since PCHIP captures the persistent pattern of the data, this process also enables us to generate a new sequence of solutions beyond the last one produced by the Lanczos algorithm. Of course, we know that extrapolation will not produce many good points. Consequently, we choose the integer s such that the residual norms of the iterates generated by this process, $\mathbf{x}_{k+1}, \mathbf{x}_{k+2}, \ldots, \mathbf{x}_s$ are small enough. It is assumed that these iterates replace the "missing" iterates not generated by the Lanczos-type algorithm due to breakdown. This is basically what EIEMLA does. However, s may be very close to k, in which case the X_s may not have a residual norm small enough to achieve convergence. This, therefore, calls for a more robust approach which is that of restarting.

3.1. Restarting EIEMLA. Restarting from a point generated by EIEMLA has been proposed for the first time in [26]. This is an implementation of the restarting strategy from some specific points advocated in [25]. The idea, checked in [25], is that re-starting algorithms of the Lanczos-type with better points results in better approximate solutions. Since EIEMLA generates approximate solutions which are always better than the iterates generated by the Lanczos-type algorithm itself up to then, these points are, therefore, good restarting points. This idea is illustrated in Figure 1, while the restarting approach is described as Algorithm 1, [26].

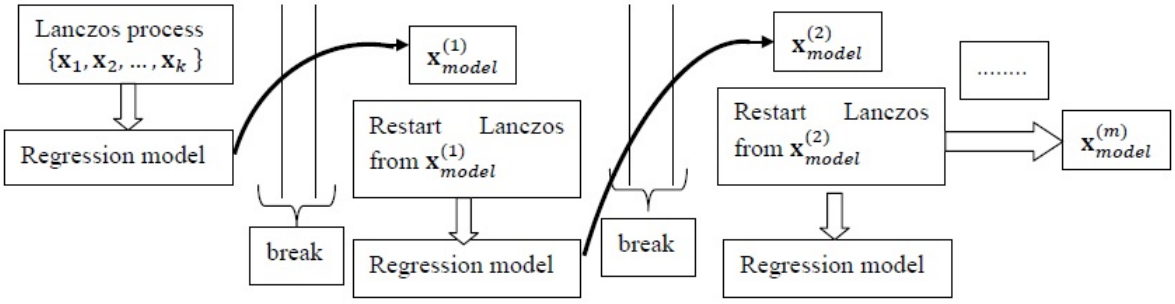

Figure 1. The process of REIEMLA on SLE's

Algorithm 1 REIEMLA

1: Initialization. Choose $\mathbf{x}_0$ and $\mathbf{y}$. Set $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, $\mathbf{y}_0 = \mathbf{y}$, and $\mathbf{z}_0 = \mathbf{r}_0$.
2: Fix the number of iterations to, say k, and the tolerance, ε, to $1E-13$.
3: Run EIEMLA for k iterations. Obtain a sequence of iterates $\{\mathbf{x}_{k+1}, \mathbf{x}_{k+2}, \ldots, \mathbf{x}_s\}$, where $s \geq> k+1$, and calculate the residual norms of these iterates.
4: Compute the minimum of the residual norms, call it $\|\mathbf{r}_{model}\|$.
5: **if** $\|\mathbf{r}_{model}\| \leq \varepsilon$ **then**
6: The solution is obtained, and it is the iterate which is associated with this residual norm, call it $\mathbf{x}_{model}$.
7: Stop.
8: **else**
9: Initialize the algorithm with

$$\begin{aligned} \mathbf{x}_0 &= \mathbf{x}_{model} \\ \mathbf{y} &= \mathbf{b} - A\mathbf{x}_0 \end{aligned}$$

10: Go to 3.
11: **end if**
12: Take $\mathbf{x}_{model}$ as the approximate solution.
13: Stop.

4. Running EIEMLA and REIEMLA in Parallel Matlab

We are concerned with the stability of the new approach when solving large scale problems (up to 1000000 variables). To run EIEMLA using the cloud computing service, the code should first be parallelized. Often parallelisation is done by hand by the user as in [30]. Here, the pain of parallelisation is taken away by the parallel environment. In Matlab, it is achieved using the *parfor*-loop function. This is illustrated in Fig. 2.

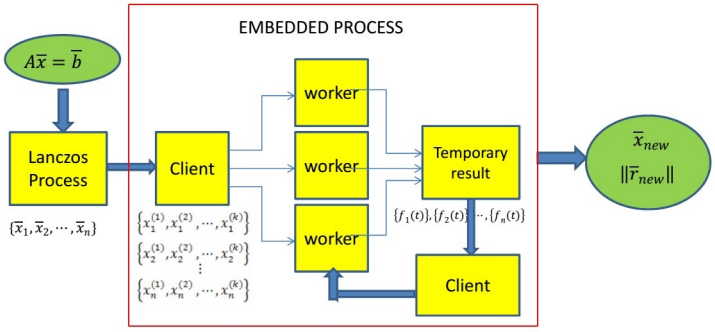

Figure 2. The embedded process in Lanczos algorithms

First, the system $A\mathbf{x} = \mathbf{b}$ is processed by Lanczos algorithm to generate a sequence $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k\}$, of approximate solutions. The sequence is then used as an input to the embedded process. In the client box, the re-arranged sequence is sent to the workers where the interpolation and extrapolation of the sequence by using PCHIP, [18], is carried out. There are a number of data sets that need to be interpolated in this stage. As said earlier, the order of computations is not preset. The amount of processing carried out on a given worker depends on the speed of the processor and the load balancing implemented by the master processor. After the whole embedded process is finished, the new approximate solution and the corresponding residual norm are produced and then sent back to the client as the final output.

4.1. Numerical Results. This section is an account of the computational experiments carried out and the numerical results obtained with the implementation of EIEMLA in a parallel environment. The comparison of the computation times needed when using *for*-loop and *parfor*-loop is presented. We also present results obtained by restarting EIEMLA (REIEMLA) in a parallel environment. We particularly focus on comparing the performance of REIEMLA, in terms of CPU time, when run on a parallel machine and on a sequential one. The test problems are solved under MATLAB 2012b on Unix0 system provided by the University of Essex which includes hardware that consists of 4 x AMD Opteron(tm) processors with 2.20 GHz speed and 48 cores, 128 GB RAM, and a 1000 Mbps ethernet interface. The Matlab PCT license is available in this system with a maximum of 8 local workers. For running the sequential algorithm, we used Matlab 2013a on a machine with 12 GB RAM.

Table 1 presents the computation time of EIEMLA in both parallel and serial environments. Several problems of dimensions ranging from 10000 to 1000000 were solved. As we can see here, the table consists of 5 columns each of which presents respectively the dimensions of the problems, the residual norms, the computation time of EIEM Orthodir with *for*-loops and *parfor*-loops, and the speed up achieved. The speed up is calculated by taking the ratio of EIEM Orthodir with *for*-loops CPU times and those of *parfor*-loops.

In general, the use of *parfor*-loops is able to reduce the execution time significantly. To solve SLE's with dimensions 10000, for instance, EIEMLA with *parfor*-loops runs four times faster than with *for*-loops. In addition, for dimensions 20000 and 30000, the speed up is 5 fold. For dimensions 40000 to 1000000, the parallel program is 5 times, sometimes 6 times, faster than the sequential one. These comparisons are clearly seen in Fig. 3.

4.2. Running REIEMLA on Parallel Platforms. To run REIEMLA in parallel, the *parfor*-loop is used as a means to parallelize our codes. Comparisons are on computation times of the algorithms with *parfor*-loop and without it, i.e. by just sticking with *for*-loops of the sequential code. Here again, problems are ranging from 100000 to 1000000 dimensions.

It can be seen in Table 2, that in general, REIEMLA runs in parallel significantly faster than sequentially. For instance, speed up is 4.04 for dimensions 100000, meaning REIEMLA solved the problems in a parallel environment 4 times

Table 1. Comparison of EIEMLA in parallel and sequential environments.

Dim	Res.Norm	Processing Time (sec.)		Speed Up
n	$\|\mathbf{r}_{model}\|$	Sequential	Parallel	x(times)
10000	4.0804	5.2229	1.2959	4.03
20000	2.0776	10.5466	2.2446	4.69
30000	14.4011	15.8540	3.2166	4.92
40000	6.7584	20.8902	3.9129	5.34
50000	3.3782	26.2809	4.9604	5.29
60000	12.3680	31.5569	5.8258	5.42
70000	5.1140	36.7257	6.8410	5.37
80000	1.3536	41.9938	7.8136	5.37
90000	5.8349	47.4259	8.8534	5.36
100000	1.0606	52.5293	9.7401	5.39
200000	5.0090	105.7368	19.2961	5.48
300000	30.4925	157.4330	28.4763	5.53
400000	30.8407	210.8508	38.6017	5.46
500000	44.7364	262.4477	49.2533	5.33
600000	71.9335	316.1252	56.3174	5.61
700000	62.2338	368.3247	65.9906	5.58
800000	16.7394	422.8909	74.7826	5.65
900000	34.0717	476.6024	83.9225	5.67
1000000	18.1782	549.0482	101.2656	5.42

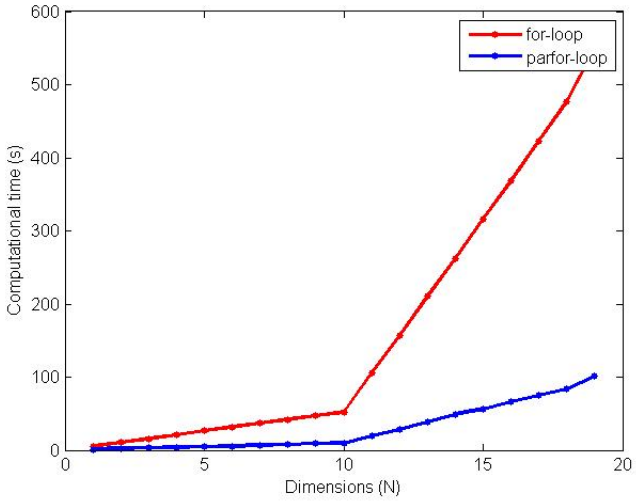

Figure 3. Comparison of *for*-loop and *parfor*-loop execution speeds

faster than restarting in a sequential environment. Also, when solving 400000 and 500000 variables, REIEMLA with *parfor*-loops was about 7 times and 6 times faster respectively than the sequential code.

5. Running EIEMLA and REIEMLA on a Cloud Computing Platform

Cloud computing is a new way of providing computing power on demand. It is potentially the future of computing since the current ownership of computers

Table 2. Comparison of REIEMLA in parallel and in sequential environments

Dim	Res.Norms	Processing Time (sec.)		Speed up
n	$\|\mathbf{r}_{model}\|$	Sequential	Parallel	x (times)
100000	$6.7572E-14$	$3.9749E+02$	98.3595	4.04
200000	$7.3917E-14$	$9.0306E+02$	$2.3294E+02$	3.88
300000	$5.7715E-14$	$1.0328E+03$	$3.6675E+02$	2.82
400000	$6.2753E-14$	$3.3668E+04$	$4.9720E+02$	6.77
500000	$7.5120E-14$	$3.5596E+03$	$5.5249E+02$	6.44
600000	$6.9701E-14$	$2.8829E+03$	$7.2046E+02$	4.00
700000	$9.6797E-14$	$4.0071E+03$	$8.8016E+02$	4.55
800000	$9.0274E-14$	$4.5529E+03$	$9.8848E+02$	4.61
900000	$7.8603E-14$	$5.5275E+03$	$1.1893E+03$	4.65
1000000	$7.8631E-14$	$6.1636E+03$	$1.2373E+03$	4.98

means that a lot of the computing power is really wasted while we pay a lot to acquire it and to maintain it, [24].

There are several cloud providers, such as Amazon EC2, Google Cloud, and Microsoft Azure, which generally provide three services, including infrastructure as a service (IIaS), the platform as a service (PaS), and software as a service (SaS), [10, 21]. Furthermore, there are different types of cloud platforms that we can subscribe to: (1) public cloud, (2) private cloud, (3) community cloud, and (4) hybrid cloud, [21, 24].

5.1. An Exemplar Cloud Computing Provider: The Domino Data Lab. Domino Data Lab, founded by Nick Elprin in 2014, is a cloud service that allows us to run R, Python, and Matlab codes, [**?**]. Its service is a platform-as-a-service (PaaS) for data analysis, to provide equipments for a larger group of users which has typically been inaccessible to people without engineering abilities. Domino runs R code (or Python, Julia, Matlab, and more) on the cloud without any set-up or configuration. It also handles Amazon Machine Images (AMI) and package management, job distribution and secure data transfer. For other advantages of Domino can be read in [13].

5.2. EIEMLA on Domino Cloud Platform. Domino has several options of hardware, from 1 GB RAM and 1 core only, to the XX large which contains 60 GB RAM and 32 cores. If the users need more RAM than those available the Domino team will set up a special hardware, the so-called "Custome hardware". In this study, we used the X-large and the XX-large hardware which respectively contain 16 and 32 cores with 30 and 60 GB RAM. We compared with Unix2 available at the university of Essex super computer with 48 cores and 256 GB shared RAM.

5.2.1. *EIEMLA on Domino Cloud : Numerical Results.* The results are recorded in Tables 3 and 4. Note, Speed up 1* is the ratio of the Domino Cloud with 16 cores CPU time and the CPU time of the local parallel machine; Speed 2* is the ratio of Domino Cloud with 32 cores CPU time and that of this same local parallel machine. This platform is the Unix2 machine of the University of Essex which supports Matlab. The way we access it is similar to the way we access the Domino cloud platform or any cloud providers for that matter. Here we present experimental results comparing performance of EIEM Orthodir on this local machine and on the Domino cloud computing.

Table 3. Performance of EIEMLA on the Cloud when solving SLEs with $\delta = 0.2$

Dim	Computational Time (seconds)	
n	Domino Cloud (16 cores)	Domino Cloud (32 cores)
100000	61.234	64.613
200000	$1.2294E+02$	$1.3160E+02$
300000	$1.8518E+02$	$1.9988E+02$
400000	$1.9288E+02$	$2.0844E+02$
500000	$3.0886E+02$	$3.1468E+02$
600000	$3.6125E+02$	$4.1542E+02$
700000	$4.4078E+02$	$4.4470E+02$
800000	$4.9064E+02$	$4.9886E+02$
900000	$5.4645E+02$	$4.8818E+02$
1000000	$5.9574E+02$	$5.6232E+02$

Table 4. CPU times of EIEMLA on the local platform and on the Domino Cloud

Dim	Local Platform (8 workers)	Speed up 1*	Speed up 2*
n	Computational Time (seconds)	x(times)	x(times)
100000	9.7401	6.29	6.63
200000	19.2961	6.37	6.82
300000	28.4763	6.5	7.02
400000	38.6017	4.9	5.39
500000	49.2533	6.27	6.39
600000	56.3174	6.41	7.38
700000	65.9906	6.68	6.74
800000	74.7826	6.56	6.67
900000	83.9225	6.51	5.82
1000000	101.2656	5.88	5.55

As can be seen in Table 3, in most cases, the processing time of 32 cores on Domino cloud is slower than 16 cores. This appears, for instance, on dimensions ranging from 100000, to 800000. For problems of dimensions 900000 and 1000000, however, the 32 cores is slightly faster than the 16 cores. This means that more processors do not necessarily translate into performance because of many factors including communication costs.

Interestingly, the local platform is more efficient than the Domino Cloud, with both 16 cores and 32 cores, as can be seen in Table 4. The processing time in the local machine is consistently less than that of the Domino Cloud. For instance, when solving 100000 dimensional problems, the execution time of the EIEM Orthodir on the university machine is 6.29x faster than the processing time on the Domino cloud with 16 cores. This factor is even bigger when we used 32 cores on Domino cloud; it is 6.6x faster. Speed up 1 and speed up 2, however, fell to 4.9 and 5.39 respectively, when solving 400000 problems. The rest of the results show the same trend.

5.3. REIEMLA on the Domino Cloud: Numerical Results. Looking at Table 5, overall, the trend is similar to that of the previous section; the execution time on the University Cloud is consistently less than that on Domino Cloud. One thing to highlight is that, the 32 cores on Domino Cloud seems to be slower than the 16 cores in some cases, while in some others it is faster. For instance, for dimensions ranging from 50000 to 80000, 100000, 300000, and 400000, the 32 cores

Table 5. Performance of Parallel REIEMLA on SLEs with $\delta = 0.2$

Dim	Computational Time (sec.)			Speed1*	Speed2*
n	Unix 0(8 nodes)	Cloud(16 cores)	Cloud(32 cores)	x(times)	x(times)
50000	49.9953	$2.48E+02$	$2.79E+02$	4.96	5.58
60000	66.3108	$2.99E+02$	$3.37E+02$	4.51	5.08
70000	75.5876	$3.17E+02$	$3.56E+02$	4.19	4.71
80000	79.946	$4.02E+02$	$4.46E+02$	5.03	5.58
90000	100.6326	$4.75E+02$	$4.65E+02$	4.72	4.62
100000	98.3595	$4.67E+02$	$5.11E+02$	4.75	5.19
200000	$2.3294E+02$	$1.052E+03$	$9.97E+02$	4.52	4.28
300000	$3.6675E+02$	$1.544E+03$	$1.650E+03$	4.21	4.49
400000	$4.9720E+02$	$2.325E+03$	$2.409E+03$	4.68	4.85
500000	$5.5249E+02$	$2.866E+03$	$2.650E+03$	5.19	4.79

is slower than the 16 cores. However, for dimensions 90000, 200000, and 500000, the 32 cores is faster than the 16 cores. So far, we do not have enough evidence to explain why this is the case, although communications overheads, and sharing of the platform with other users may be the reason. We also suspect that our code is accessing some shared resource (e.g., a global matrix). So, it is possible that the underlying operating system is doing some locking to prevent multiple threads from accessing that resource at the same time. If that is really happening, then more threads could slow things down, [12].

6. Conclusion

We have introduced a novel approach to addressing the inherent fragility of Lanczos-type algorithms by way of interpolating and extrapolating sequences of iterates generated by the Lanczos-type algorithm used prior to it breaking down. This approach, namely EIEMLA, although robust, does not achieve the expected convergence. A restarting strategy is thus implemented into it leading to REIEMLA which is very robust and efficient. Because of the ubiquitous and common nature of SLEs, we set out to solve problems of large scale to test our approach. Moreover, given now the advent of cloud computing, we decided to implement and run both EIEMLA and REIEMLA on a standard parallel processing platform as provided by the University of Essex where this work has been carried out and also on a commercial cloud computing platform, namely Domino Data Lab. Very good computational results have been obtained showing that the approach does indeed address the issue of breakdown in Lanczos-type algorithms and does it in an efficient way. Moreover, it can be run in parallel on the most up-to-date platforms quite straightforwardly.

There is a slight problem, however. At the moment we have to choose k, the length of the sequence of iterates generated by the algorithm used in an arbitrary fashion. This is because we really do not know when the Lanczos-type algorithm is going to break down. If we wait until the algorithm breaks down then we will have to pick up the pieces, so to speak, reset everything and try again which can be costly. If we stop the process too early, then we would have wasted potentially a number of useful iterations. It is therefore worthwhile to investigate preemptive restarting which can deliver the appropriate values of k. We are currently working on this. Our finding will be included in forthcoming paper.

Acknowledgments

This project is supported by BATCH I Research Grant University of Jenderal Soedirman, Indonesia, 2017. Thanks to University of Essex for providing us a massive computing systems.

References

[1] E. Ayachour *Avoiding Look-Ahead in Lanczos Method and Pade Approximation*, Applicationes Mathematica, 1999.

[2] Baheux, C. *New implementations of Lanczos method*, Journal of Computational and Applied Mathematics **57** (1-2), 3-15, 1995.

[3] Brezinski, Claude and Sadok, Hassane *Lanczos-type algorithms for solving systems of linear equations*, Applied Numerical Mathematics **11** (6), 443-473, 1993.

[4] Brezinski, C. and Zaglia, R. *A New Presentation of Orthogonal Polynomials with Applications to Their Computation*, Numerical Algorithms **1** (2), 207-221, 1991.

[5] Brezinski, C. and Zaglia, R. *Breakdowns in the Computation of Orthogonal Polynomials*, Springer, Dordrech **296**, 49-59, 1994.

[6] Brezinski, C. and Zaglia, R. and Sadok,H.*Avoiding Breakdown and Near-Breakdown in Lanczos-type Algorithms*, Numerical Algorithms **1** (2), 261-284, 1991.

[7] Brezinski, C. and Zaglia, R. and Sadok, H.*A Breakdown-Free Lanczos-type Algorithm for Solving Linear Systems*, Numerical Mathematics **63** (1), 29-38, 1992.

[8] Brezinski, C. and Zaglia, R. and Sadok, H.*New Look - Ahead Lanczos - type Algorithms for Linear Systems*, Numerical Mathematics **83**, 53-85, 2000.

[9] Brezinski, C. and Zaglia, R. and Sadok, H.*The Matrix and Polynomial Approaches to Lanczos-type Algorithms*, Journal Of Computational and Applied Mathematics **123** (1-2), 241-260, 2000.

[10] Buyya, Rajkumar and Broberg, James and Goscinski, Andrej *Cloud Computing Principles and Paradigms*, John Wiley and Sons **John Wiley and Sons, New Jersey** (1-2), 241-260, 2011.

[11] Duff, L.S. and Van der Vorst, H.A. *Developments and Trends in the Parallel Solution of Linear Systems*, Parallel Computing **25**, 1931-1970, 1999.

[12] Elprin, Nick *Private Discussion*, 11 August 2014.

[13] Elprin, Nick *Domino: A Platform-as-a-Service for Industrialized Data Analysi*, http:/www.help.dominodatalab.com/keyConcepts, 2014.

[14] Farooq, M. and Salhi, A. *TNew Recurrence Relationships between Orthogonal Polynomials which Lead to New Lanczos-type Algorithms*, Journal of Prime Research in Mathematics **8**, 61-75, 20012.

[15] Farooq, M. and Salhi, A. *A Preemptive Restarting Approach to Beating Inherent Instability*, Iranian Journal of Science and Technology Transaction a Science **37**, (Special Issue) 349-358, 2013.

[16] Farooq, M. and Salhi, A. *A Switching Approach to Avoid Breakdown in Lanczos-type Algorithms*, Applied Mathematics and Information Sciences **8**, (5) 2161-2169, 2014.

[17] Fritsch, F. N. and Butland, J. *A Method for Constructing Local Monotone Piecewise Cubic Interpolants*, SIAM J. Sci. Stat. Comput. **5**, (2) 300-304, 1984.

[18] Fritsch, F. N. and Carlson, R.E. *Monotone Piecewise Cubic Interpolation*, SIAM Journal Numerical Analysis **17**, (2) 239-248, 1980.

[19] Grave-Morris, P. *A Look-Around Lanczos Algorithm for Solving a System of Linear Equations*, Numerical Algorithm **15**, 247-274, 1997.

[20] Heller, Don *A Survey of Parallel Algorithms in Numerical Linear Algebra*, Technical report, Department of Computer Science, Carnegie Mellon University **1-1-1976**, http://repository.cmu.edu/compsci, 1976.

[21] Huth, Alexa and Cebula *The Basics of Cloud Computing*, Technical report, Produced for US-CERT, Carnegie Mellon University **www.us-cert.gov/sites/default/files/.../CloudComputingHuthCebula.pdf**, 2011.

[22] Lanczos, C. *An Iteration Method for The Solution of the Eigenvalue Problem of Linear Differential and Integral Operator*, J.Res.Natl.Bur.Stand **45**, (4), 255-282, 1958.

[23] Lanczos, C. *Solution of Systems of Linear Equations by Minimized Iterations*, J.Res.Natl.Bur.Stand **49**, (1), 33-53, 1952.

[24] Landis, C. and Blacharski, D. *Cloud Computing Made Easy*, Virtual Global Inc., 2013.

[25] Maharani, M. and Salhi, A. *Restarting from Specific Points to Cure Breakdown in Lanczos-type Algorithms*, Journal of Mathematical and Fundamental Sciences **2**, (47), 167-184, 2015.

[26] Maharani, M. and Salhi, A. and Khan,Wali *Enhanced the Stability of Lanczos-type Algorithms by Restarting The Point Generated by EIEMLA for the Solution of Systems of Linear Equations*, Science Journal Lahore **28**, (4), 3325-3335, 2016.

[27] Rajalakshami, K. *Parallel Algorithm for Solving Large Systems of Simultaneous Linear Equations*, IJCSNS **9**, (7), 276-279, 2009.

[28] Rashid, M., Crowcroft, J. *Parallel Iterative Solution Method for Large Sparse Linear Equation Systemss*, Tech. Rep. 650, **Technical report, University of Cambridge, Computer Laboratory**, October 2005.

[29] Saad, Y. *Iterative Methods for Sparse Linear Systems*, Philadelphia: Society for Industrial and Applied Mathematics, 2003.

[30] Salhi, A., Proll, L. G., Rios Insua, D. *Parallelising an Optimisation-based Framework for Sensitivity Analysis in MultiCriteria Decision Making*, Tech. Rep. 98-15, **Mathematics Department, University of Essex, UK**, 1998.

[31] Sidi, A., William Ford, F., David Smith, A. *Acceleration of Convergence of Vector Sequences*, SIAM Journal on Numerical Analysis **23**, (1), 178–196, 1986.

[32] Torp, Audun *Sparse Linear Algebra on a GPU with Applications to Flow in Porous Media*, Ph.D. thesis, **Department of Physics and Mathematics, Norwegian University of Science and Technology**, 2009.