Journal of Discrete Algorithms ••• (••••) •••–•••

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

A condensation-based application of Cramer's rule for solving large-scale linear systems

Ken Habgood *, Itamar Arel

Department of Electrical Engineering and Computer Science, The University of Tennessee, Knoxville, TN, USA

ARTICLE INFO

Article history:
Received 4 April 2010
Received in revised form 27 June 2011
Accepted 27 June 2011
Available online xxxx

Keywords:
Cramer's rule
Linear systems
Matrix condensation

ABSTRACT

State-of-the-art software packages for solving large-scale linear systems are predominantly founded on Gaussian elimination techniques (e.g. LU-decomposition). This paper presents an efficient framework for solving large-scale linear systems by means of a novel utilization of Cramer's rule. While the latter is often perceived to be impractical when considered for large systems, it is shown that the algorithm proposed retains an $\mathcal{O}(N^3)$ complexity with pragmatic forward and backward stability properties. Empirical results are provided to substantiate the stated accuracy and computational complexity claims.

1. Introduction

Systems of linear equations are central to many science and engineering application domains. Fast linear solvers generally use a form of Gaussian elimination [8], the most common of which is LU-decomposition. The latter involves a computation complexity of $W_{LU} \approx \frac{2}{3}N^3$ [11], where N denotes the number of linearly independent columns in a matrix. The factor 2 accounts for one addition and one multiplication. If only multiplications are considered, then $W_{LU} \approx \frac{N^3}{3}$, which is the operation count sometimes quoted in the literature.

In most implementations of Gaussian elimination, the row with the largest lead value and the first row are interchanged during each reduction. This is referred to as partial pivoting and facilitates the minimization of truncation errors. Historically, Cramer's rule has been considered inaccurate when compared to these methods. As this paper will discuss, the perceived inaccuracy does not originate from Cramer's rule but rather from the method utilized for obtaining determinants.

This paper revisits Cramer's rule [4] and introduces an alternative framework to the traditional LU-decomposition methods offering similar computational complexity and storage requirements. To the best of the authors' knowledge, this is the first work to demonstrate such characterization. In utilizing a technique similar to partial pivoting to calculate determinant values, the algorithm's stability properties are derived and shown to be comparable to LU-decomposition for asymmetric systems.

The rest of this paper is structured as follows. Section 2 describes the algorithm and presents its computational complexity. Section 3 discusses the stability of the algorithm including forward and backward error analysis. Section 4 presents empirical results to support the proposed complexity assertions. Finally, a brief discussion and drawn conclusions are provided in Section 5.

* Corresponding author.
E-mail addresses: habgood@eecs.utk.edu (K. Habgood), itamar@eecs.utk.edu (I. Arel).

1570-8667/$ – see front matter
doi:10.1016/j.jda.2011.06.007

Please cite this article in press as: K. Habgood, I. Arel, A condensation-based application of Cramer's rule for solving large-scale linear systems, Journal of Discrete Algorithms (2011), doi:10.1016/j.jda.2011.06.007

2. Revisiting Cramer's rule

The proposed algorithm centers on the mathematically elegant Cramer's rule, which states that the components of the solution to a linear system in the form $Ax = b$ (where A is invertible) are given by

$$x_i = \det\big(A_i(b)\big)/\det(A), \tag{1}$$

where $A_i(b)$ denotes the matrix A with its ith column replaced by b [10]. Unfortunately, when working with large matrices, Cramer's rule is generally considered impractical. This stems from the fact that the determinant values are calculated via minors. As the number of variables increases, the determinant computation becomes unwieldy [3]. The time complexity is widely quoted as $\mathcal{O}(N!)$, which would make it useless for any practical application when compared to a method like LU-decomposition at $\mathcal{O}(N^3)$. Fortunately, Cramer's rule can be realized in far lower complexity than $\mathcal{O}(N!)$. The complexity of Cramer's rule depends exclusively on the determinant calculations. If the determinants are calculated via minors, that complexity holds. In an effort to overcome this limitation, matrix condensation techniques can reduce the size of the original matrix to one that may be solved efficiently and quickly. As a result, Cramer's rule becomes $\mathcal{O}(N^3)$ similar to LU-decomposition.

The other concern with Cramer's rule pertains to the numerical instability, which is less studied [10]. A simple example put forward in [12] suggests that Cramer's rule is unsatisfactory even for 2-by-2 systems, mainly because of round error difficulties. However, that argument heavily depends on the method for obtaining the determinants. If an accurate method for evaluating determinants is used then Cramer's rule can, in fact, be numerically stable. In fact, a later paper [5] revisited the cited example and provided an example where Cramer's rule yielded a highly accurate answer while Gaussian elimination with pivoting a poor one. We thus provide a detailed treatment of the stability and accuracy aspects of the proposed determinant calculations employed by the proposed algorithm.

2.1. Chio's condensation

Chio's condensation [6] method reduces a matrix of order N to order $N-1$ when evaluating its determinant. As will be shown, repeating the procedure numerous times can reduce a large matrix to a size convenient for the application of Cramer's rule. Chio's pivotal condensation theorem is described as follows. Let $A = [a_{ij}]$ be an $N \times N$ matrix for which $a_{11} \neq 0$. Let D denote the matrix obtained by replacing elements a_{ij} not in the lead row or lead column by $\left|\begin{smallmatrix} a_{11} & a_{1j} \\ a_{i1} & a_{ij} \end{smallmatrix}\right|$, then it can be shown that $|A| = \frac{|D|}{a_{11}^{n-2}}$ [6].

Note that this process replaces each element in the original matrix with a 2×2 determinant consisting of the a_{11} element, the top value in the element's column, the first value in the element's row and the element being replaced. The calculated value of this 2×2 determinant replaces the initial $a_{i,j}$ with $a'_{i,j}$. The first column and first row are discarded, thereby reducing the original $N \times N$ matrix to an $(N-1) \times (N-1)$ matrix with an equivalent determinant. As an example, we consider the following 3×3 matrix:

$$A = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} \quad \text{and its condensed form:}$$

$$\begin{vmatrix} 0 & 0 & 0 \\ 0 & (a_{11}a_{22} - a_{21}a_{12}) & (a_{11}a_{23} - a_{21}a_{13}) \\ 0 & (a_{11}a_{32} - a_{31}a_{12}) & (a_{11}a_{33} - a_{31}a_{13}) \end{vmatrix} = \begin{vmatrix} \times & \times & \times \\ \times & a'_{22} & a'_{23} \\ \times & a'_{32} & a'_{33} \end{vmatrix}$$

Obtaining each 2×2 determinant requires two multiplications and one subtraction. However, if the value of $a_{1,1}$ is one, then only a single multiplication is required. In the example above we note that $a_{1,1}$ is used in each element as a multiplier to the matrix element, for example, the equation for the matrix element in position $(2,2)$ is $a_{11}a_{22} - a_{21}a_{12}$. If in this situation $a_{11} = 1$, then the equation changes to $a_{22} - a_{21}a_{12}$. This holds true for every element in the matrix. Therefore for each condensation step k, if $a_{kk} = 1$ then $(N-k)^2$ multiplications are removed.

In order to guarantee $a_{kk} = 1$, an entire row or column must be divided by a_{kk}. This value would need to be stored because the determinant value calculated by Chio's condensation would be reduced by this factor. To find the true value at the end of the condensation, the calculated answer would need to be multiplied by each a_{kk} that was factored out. Multiplying all of these values over numerous condensation steps would result in an extremely large number that would exceed the floating point range of most computers. This is where the elegance of Cramer's rule is exploited. Cramer's rule determines each variable by a ratio of determinants, $x_i = \det(A_i(b))/\det(A)$. Given that both determinants are from the same condensation line, they both are reduced by the same a_{kk} values. The a_{kk} values factored out during Chio's condensation cancel during the application of Cramer's rule. This allows the algorithm to simply discard the a_{kk} values in the final computations. The actual determinant values are not correct, however the ratio evaluated at the core of Cramer's rule remains correct.

Please cite this article in press as: K. Habgood, I. Arel, A condensation-based application of Cramer's rule for solving large-scale linear systems, Journal of Discrete Algorithms (2011), doi:10.1016/j.jda.2011.06.007

The cost of using Chio's condensation is equivalent to computing $(N-k)^2$ 2×2 determinants and $(N-k)$ divisions to create $a_{kk}=1$. Hence, the computational effort required to reduce an $N\times N$ matrix to a 1×1 matrix is $\mathcal{O}(N^3)$, since

$$\sum_{k=1}^{N-1} 2(N-k)^2+(N-k)=\frac{2N^3}{3}-\frac{N^2}{2}-\frac{N}{6}-\sim\mathcal{O}\big(N^3\big). \tag{2}$$

Combined with Cramer's rule, this process can yield the determinant to find a single variable. In order to find all N variables, this would need to be repeated N times, suggesting that the resulting work would amount to $\mathcal{O}(N^4)$. We next describe an efficient way of overcoming the need for $\mathcal{O}(N^4)$ computations by sharing information pertaining to each variable.

2.2. Matrix mirroring

The overarching goal of the proposed approach is to obtain an algorithm with $\mathcal{O}(N^3)$ complexity and low storage requirement overhead. As discussed above, Chio's condensation and Cramer's rule provide an elegant solution with $\mathcal{O}(N^4)$ complexity. In order to retain $\mathcal{O}(N^3)$ computational complexity, it is necessary to reuse some of the intermediate calculations performed by prior condensation steps. This is achieved by constructing a binary, tree-based data flow in which the algorithm mirrors the matrix at critical points during the condensation process, as detailed next.

A matrix A and a vector of constants b are passed as arguments to the algorithm. The latter begins by appending b to A creating an augmented matrix. All calculations performed on this matrix are also performed on b. Normal utilization of Cramer's rule would involve substitution of the column corresponding to a variable with the vector b, however the proposed algorithm introduces a delay in such substitution such that multiple variables can be solved utilizing one line of Chio's condensation. In order to delay the column replacement, b must be subject to the same condensation manipulations that would occur had it already been in place. This serves as the motivation for appending b to the matrix during condensation.

The condensation method removes information associated with discarded columns, which suggests that the variables associated with those columns cannot be computed once condensed. For this reason, a mirror of the matrix is created each time the matrix size is halved. The mirrored matrix is identical to the original except the order of its columns is reversed. For example, the first and last column are swapped, the second and second to last column are swapped, and so on. A simple 3×3 matrix mirroring operation would be:

$$\begin{vmatrix} a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}\\ a_{31} & a_{32} & a_{33}\end{vmatrix} \rightarrow \text{mirrored} \rightarrow \begin{vmatrix} a_{13} & a_{12} & -a_{11}\\ a_{23} & a_{22} & -a_{21}\\ a_{33} & a_{32} & -a_{31}\end{vmatrix}.$$

In the example above, the third column of the mirror is negated, which is performed in order to retain the correct value of the determinant. Any exchange of columns or rows requires the negation of one to preserve the correct determinant value. As discussed in more detail below, this negation is not necessary to arrive at the correct answer, but is applied for consistency.

Following the mirroring, each matrix is assigned half of the variables. The original matrix can solve for the latter half while the mirrored matrix solves for the first half of the variables. In the example above, there are three variables: x_1, x_2, x_3. The original matrix could solve for x_2, x_3, and the mirrored matrix would provide x_1. Each matrix uses condensation to yield a reduced matrix with size at least equal to the number of variables it's responsible for. For the case of a 3×3 matrix, we have the pair of matrices:

$$\underbrace{\begin{vmatrix} \times & \times & \times\\ \times & a'_{22} & a'_{23}\\ \times & a'_{32} & a'_{33}\end{vmatrix}}_{\text{original: } x_2,x_3} \quad \underbrace{\begin{vmatrix} \times & \times & \times\\ \times & a'_{22} & a'_{21}\\ \times & a'_{32} & a'_{31}\end{vmatrix}}_{\text{mirrored: } x_1}.$$

Once this stage is reduced, the algorithm either solves for the variables using Cramer's rule or mirrors the matrix and continues with further condensation. A process flow depicting the proposed framework is illustrated in Fig. 1.

2.3. Extended condensation

Chio's condensation reduces the matrix by one order per repetition. Such an operation is referred to here as a condensation step of size one. It's possible to reduce the matrix by more than one order during each step. Carrying out the condensation one stage further, with leading pivot $\left|\begin{smallmatrix} a_{11} & a_{12}\\ a_{21} & a_{22}\end{smallmatrix}\right|$ (assumed to be non-zero), we have [2]

Please cite this article in press as: K. Habgood, I. Arel, A condensation-based application of Cramer's rule for solving large-scale linear systems, Journal of Discrete Algorithms (2011), doi:10.1016/j.jda.2011.06.007

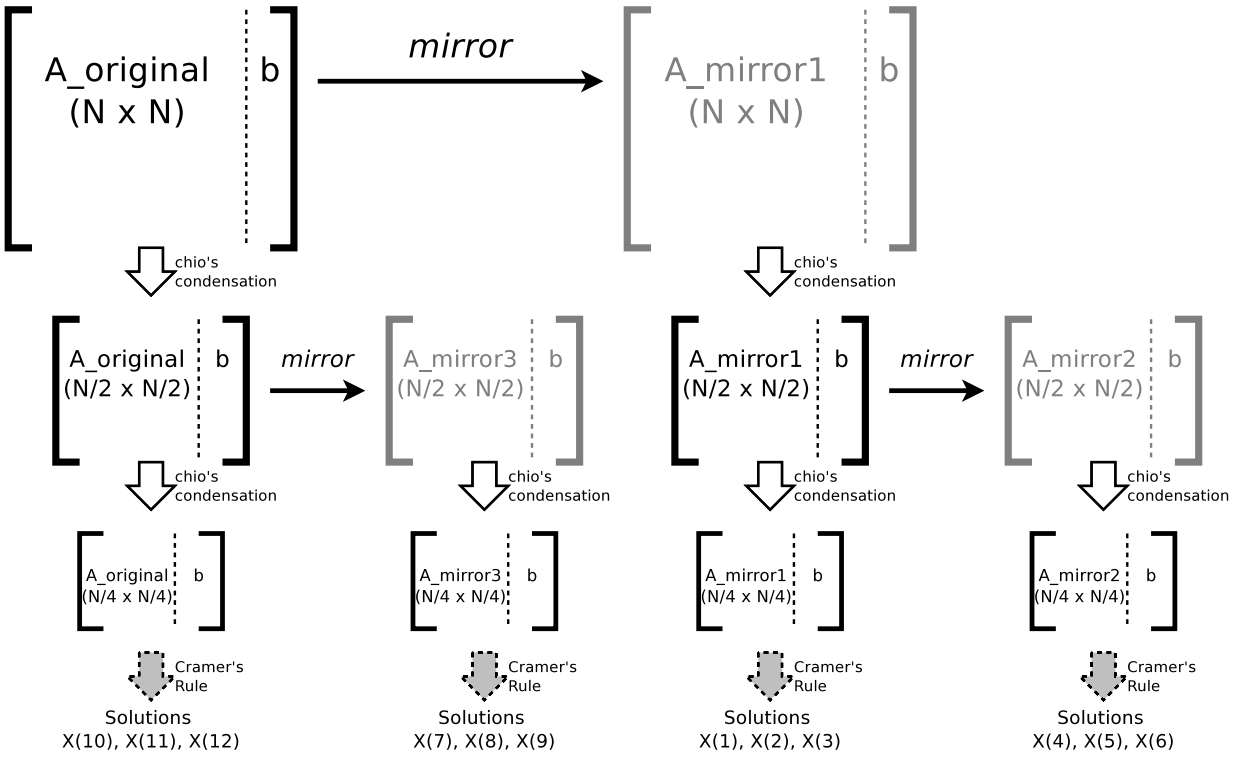

Fig. 1. Tree architecture applied to solving a linear system using the proposed algorithm.

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{vmatrix} = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}^{-1} \times \begin{vmatrix} \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{12} & a_{14} \\ a_{21} & a_{22} & a_{24} \\ a_{31} & a_{32} & a_{34} \end{vmatrix} \\ \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{41} & a_{42} & a_{43} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{12} & a_{14} \\ a_{21} & a_{22} & a_{24} \\ a_{41} & a_{42} & a_{44} \end{vmatrix} \end{vmatrix}, \tag{3}$$

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{vmatrix} = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}^{-1} \times \begin{vmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & a'_{33} & a'_{34} \\ \times & \times & a'_{43} & a'_{44} \end{vmatrix}. \tag{4}$$

In this case, each of the matrix elements $\{a_{33}, a_{34}, a_{43}, a_{44}\}$ is replaced by a 3×3 determinant instead of a 2×2 determinant. This delivers a drastic reduction in the number of repetitions needed to condense a matrix. Moreover, a portion of each minor is repeated, namely the $\left|\begin{smallmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{smallmatrix}\right|$ component. Such calculation can be performed once and then reused multiple times during the condensation process. Letting M denote the size of the condensation, and using the example above, $M = 2$ while for the basic Chio's condensation technique $M = 1$. As an example, a 6×6 matrix could be reduced to a 2×2 matrix in only two condensation steps, whereas it would take four traversals of the matrix to arrive at a 2×2 matrix if $M = 1$. The trade-off is a larger determinant to calculate for each condensed matrix element. Instead of having 2×2 determinants to calculate, 3×3 determinants are needed, suggesting a net gain of zero.

However, the advantage of this formulation is that larger determinants in the same row or column share a larger number of the minors. The determinant minors can be calculated at the beginning of each row and then reused for every element in that row. This reduces the number of operations required for each element with a small penalty at the outset of each row. In practical computer implementation, this also involves fewer memory access operations, thus resulting in higher overall execution speed.

Pivoting in the case of $M > 1$ requires identifying a lead determinant that is not small. As with pivoting for LU-decomposition, ideally the largest possible lead determinant would be moved into the top left portion of the matrix. Unfortunately, this severely compromises the computational complexity, since an exhaustive search for the largest lead determinant is impractical. Instead, a heuristic method should be employed to select a relatively large lead determinant when compared to the alternatives.

Please cite this article in press as: K. Habgood, I. Arel, A condensation-based application of Cramer's rule for solving large-scale linear systems, Journal of Discrete Algorithms (2011), doi:10.1016/j.jda.2011.06.007

Algorithm 1 Extended condensation.

```
{A[][] = current matrix, N = size of current matrix}
{mirrorsize = unknowns for this matrix to solve for}
while (N − M) > mirrorsize do
  lead_determinant = CalculateMinor(A[][], M + 1, M + 1)
  if lead_determinant = 0 then
    return(error)
  end if
  A[1:N][1] = A[1:N][1] / lead_determinant;
  {calculate the minors that are common}
  for i = 1 to M step by 1 do
    for j = 1 to M step by 1 do
      {each minor will exclude one row & col}
      reusableminor[i][j] = CalculateMinor(A[][], i, j);
    end for
  end for
  for row = (M + 1) to (N + 1) step by 1 do
    {find the lead minors for this row}
    for i = 1 to M step by 1 do
      Set leadminor[i] = 0;
      for j = 1 to M step by 1 do
        leadminor[i] = leadminor[i] + (−1)^(j−1) A[row][j] × reusableminor[i][j]
      end for
    end for
    {Core Loop; find the M × M determinant for each A[][] item}
    for col = (M + 1) to (N + 1) step by 1 do
      for i = 1 to M step by 1 do
        {calculate MxM determinant}
        A[row][col] = A[row][col] + (−1)^j leadminor[i] × A[i][col]
      end for
    end for
  end for
  {Reduce matrix size by condensation step size}
  N = N − M;
end while
if N has reached Cramer's rule size (typically 4) then
  {solve for the subset of unknowns assigned}
  x[] = CramersRule(A[][]);
else {recursive call to continue condensation}
  A_mirror[][] = Mirror(A[][])
  Recursively call Algorithm (A_mirror[][], N, mirrorsize/2)
  Recursively call Algorithm (A[][], N, mirrorsize/2)
end if
```

The pseudocode in Algorithm 1 details a basic implementation of the proposed algorithm using extended condensation. The number of rows and columns condensed during each pass of the algorithm is represented by the variable M, referred to earlier as the condensation step size. The original matrix is passed to the algorithm along with the right-hand side vector in A, which is an $N \times (N+1)$ matrix. The *mirrorsize* variable represents the number of variables a particular matrix solves for. In other words, it reflects the smallest size that a matrix can be condensed to before it must be mirrored. The original matrix, A, has a $mirrorsize = N$ since it solves for all N variables. It should be noted that this will result in bypassing the while loop completely at the initial call of the algorithm. Prior to performing any condensations, the original matrix is mirrored. The first mirror will have $mirrorsize = \frac{N}{2}$, since it only has to solve for half of the variables. After this mirror has been created, the algorithm will begin the condensation identified within the while loop.

Three external functions assist the pseudocode: *CalculateMinor*, *CramersRule* and *Mirror*. *CalculateMinor* finds an $M \times M$ determinant from the matrix passed as an argument. The two additional arguments passed identify the row and column that should be excluded from the determinant calculation. For example, $CalculateMinor(A[][], 2, 3)$ would find the $M \times M$ determinant from the top left portion of matrix $A[][]$, whereby row 2 and column 3 are excluded. The method would return the top left $M \times M$ determinant of $A[][]$ without excluding any rows or columns by calling $CalculateMinor(A[][], M+1, M+1)$, since an $M+1$ value excludes a column beyond the M rows and M columns used to calculate the determinant. This is used in the algorithm to find the lead determinant at each condensation step. When $M = 1$, the method simply returns the value at $A[1][1]$.

The *CramersRule* method solves for the variables associated with that particular matrix using Cramer's rule. The method replaces a particular column with the condensed solution vector, b, finds the determinant, and divides it by the determinant of the condensed matrix to find each variable. The method then replaces the other columns until all variables for that particular phase are calculated. The *Mirror* function creates a mirror of the given matrix as described in Section 2.2.

The arrays labeled *reusable_minor* and *lead_minor* maintain the pre-calculated values discussed earlier. The array *reusable_minor* is populated once per condensation step and holds M^2 minors that will be used at the outset of each row. The latter will then populate the *lead_minor* array from those reusable minors. The *lead_minor* array holds the M minors

Please cite this article in press as: K. Habgood, I. Arel, A condensation-based application of Cramer's rule for solving large-scale linear systems, Journal of Discrete Algorithms (2011), doi:10.1016/j.jda.2011.06.007

needed for each matrix element in a row. Hence, the *lead_minor* array will be repopulated $N - M$ times per condensation step.

The algorithm divides the entire first column by the top left $M \times M$ determinant value. This is performed for the same reason that the first column of the matrix was divided by the a_{11} element in the original Chio's condensation formulation. Dividing the first row by that determinant value causes the lead determinant calculation to retain a value of one during the condensation. This results in a '1' being multiplied by the $a_{i,j}$ element at the beginning of every calculation, thus saving one multiplication operation per each element.

2.4. Computation complexity

As illustrated in the pseudocode, the core loop of the algorithm involves the calculation of the $M \times M$ determinants for each element of the matrix during condensation. Within the algorithm, each $M \times M$ determinant requires M multiplications and M additions/subtractions. Normally, this would necessitate the standard computational workload to calculate a determinant, i.e. $\frac{2}{3}M^3$, using a method such as Gaussian elimination. However, the reuse of the determinant minors described earlier reduces the effort to $2M$ operations within the core loop.

An additional workload outside the core loop is required since M^2 minors must be pre-calculated before Chio's condensation commences. Assuming the same $\frac{2}{3}M^3$ workload using Gaussian elimination to find a determinant and repetition of this at each of the $\frac{N}{M}$ condensation steps, yields an overhead of

$$\frac{N}{M} \times M^2 \times \frac{2}{3}M^3 = \frac{2M^4N}{3}. \tag{5}$$

In situations where $M \ll N$, this effort is insignificant, although with larger values of M relative to N, it becomes non-negligible.

The optimal size of M occurs when there's a balance between pre-calculation done outside the core loop and the savings during each iteration. In order to reuse intermediate calculation results, a number of determinant minors must be evaluated in advance of each condensation step. These reusable minors save time during the core loops of the algorithm, but do not utilize the most efficient method. If a large number of minors are required prior to each condensation, their additional computation annuls the savings obtained within the core iterations.

The optimal size of M is thus calculated by setting the derivative of the full complexity formula, $\frac{2}{3}N^3 - MN^2 + \frac{N^2}{M} + \frac{2}{3}M^4N + M^2N$, with respect to M to zero. This reduces to $\frac{8}{3}M^5 + 2M^3 = NM^2 + N$, suggesting an optimal value of $\sqrt[3]{\frac{3}{8}N}$ for M. As an example, the optimal point for a 1000×1000 matrix is ≈ 7.22. Empirical results indicate that the shortest execution time for a 1000×1000 matrix was achieved when $M = 8$, supporting the theoretical result.

In order to condense a matrix from $N \times N$ to $(N - M) \times (N - M)$, the core calculation is repeated $(N - M)^2$ times. The algorithm requires N/M condensation steps to reduce the matrix completely and solve using Cramer's rule. In terms of operations, this equates to

$$\begin{aligned}
\gamma &= \sum_{k=1}^{N/M} 2M(N - kM)^2 \\
&= 2M \sum_{k=1}^{N/M} \left(N^2 - 2NMk + M^2k^2\right) \\
&= 2M\left(\frac{N}{M}N^2 - 2NM\left(\frac{\frac{N}{M}(\frac{N}{M}+1)}{2}\right) + M^2\left(\frac{\frac{N}{M}(\frac{N}{M}+1)(\frac{2N}{M}+1)}{6}\right)\right) \\
&= \frac{2}{3}N^3 - MN^2 + \frac{M^2N}{3},
\end{aligned} \tag{6}$$

resulting in a computational complexity, γ, of $\frac{2}{3}N^3$ to obtain a single variable solution.

Mirroring occurs with the initial matrix and then each time a matrix is reduced in half. An $N \times N$ matrix is mirrored when it reaches the size of $\frac{N}{2} \times \frac{N}{2}$. Once the matrix is mirrored, there is double the work. In other words, two $\frac{N}{2} \times \frac{N}{2}$ matrices each require a condensation, where previously there was only one. However, the amount of work for two matrices of half the size is much lower than that of one $N \times N$ matrix, which avoids the $O(N^4)$ growth pattern in computations. This is due to the $O(N^3)$ nature of the condensation process.

Since mirroring occurs each time the matrix is reduced to half, $\log_2 N$ matrices remain when the algorithm concludes. The work associated with each of these mirrored matrices needs to be included in the overall computation load estimate. The addition of the mirrors follows a geometric series resulting in roughly 2.5 times the original workload, which leads to a computational complexity of $\frac{5}{3}N^3$ when ignoring the lower order terms.

The full computational complexity is the combination of the work involved in reducing the original matrix, γ, and that of reducing the mirrors generated by the algorithm. Hence, the total complexity can be expressed as follows:

Please cite this article in press as: K. Habgood, I. Arel, A condensation-based application of Cramer's rule for solving large-scale linear systems, Journal of Discrete Algorithms (2011), doi:10.1016/j.jda.2011.06.007

$$\gamma + \sum_{k=0}^{\log_2 N} 2^k \left(\frac{2}{3} \left(\frac{N}{2^k} \right)^3 - M \left(\frac{N}{2^k} \right)^2 + \frac{M^2}{3} \left(\frac{N}{2^k} \right) \right). \tag{7}$$

The latter summation can be simplified using the geometric series equivalence $\sum_{k=0}^{n-1} ar^k = a\frac{1-r^n}{1-r}$ [1], which when ignoring the lower order terms reduces to:

$$\gamma + \frac{8}{9}N^3 = \frac{14N^3}{9} \approx \frac{5}{3}N^3. \tag{8}$$

2.4.1. Mirroring considerations and related memory requirements

Eq. (8) expresses the computational complexity assuming a split into two matrices each time the algorithm performs mirroring. One may consider a scenario in which the algorithm creates more than two matrices during each mirroring step. In the general case, the computational complexity is given by

$$\gamma + \sum_{k=0}^{\log_S N} (S-1)S^k \left(\frac{2}{3} \left(\frac{N}{S^k} \right)^3 - M \left(\frac{N}{S^k} \right)^2 + \frac{M^2}{3} \left(\frac{N}{S^k} \right) \right) \tag{9}$$

where S denotes the number of splits. When lower order terms are ignored, this yields

$$\gamma + \frac{2S^2}{3(S+1)}N^3. \tag{10}$$

As S increases the complexity clearly grows. The optimal number of splits is thus two, since that represents the smallest value of S that can still solve for all variables. Additional splits could facilitate more work in parallel, however they would generate significantly greater overall workload.

The memory requirement of the algorithm is $2 \times (N+1)^2$, reflecting the need for sufficient space for the original matrix and the first mirror. The rest of the algorithm can reuse that memory space. Since the memory requirement is double the amount required by typical LU-decomposition implementations and similar to LU-decomposition, the original matrix is overwritten during calculations.

3. Algorithm stability

The stability properties of the proposed algorithm are very similar to those of Gaussian Elimination techniques. Both schemes are mathematically accurate yet subject to truncation and rounding errors. As with LU-decomposition, if these errors are not accounted for, the algorithm returns poor accuracy. LU-decomposition utilizes partial or complete pivoting to minimize truncation errors. As will be shown, the proposed algorithm employs a similar technique.

Each element during a condensation step is affected by the lead determinant and the 'lead minors' discussed in the earlier section. In order to avoid truncation errors, these values should go from largest on the first condensation step to smallest on the last condensation step. This avoids a situation where matrix values are drastically reduced, causing truncation, and then significantly enlarged later, magnifying the truncation error. The easiest method to avoid this problem is by moving the rows that would generate the largest determinant value to the lead rows before each condensation step. This ensures the largest determinant values available are used.

Once the matrix is rearranged with the largest determinant in the lead, normalization occurs. Each lead value is divided by the determinant value, resulting in the lead determinant equaling unity. This not only reduces the number of floating point calculations but serves to normalize the matrix.

3.1. Backward error analysis

The backward stability analysis presented yields results similar to LU-decomposition. This coupled with empirical findings provides evidence that the algorithm yields accuracy comparable to that of LU-decomposition. As with any computer calculations, rounding errors affect the accuracy of the algorithm's solution. Backward stability analysis shows that the solution provided is the exact solution to a slightly perturbed problem. The typical notation for this concept is

$$(A+F)\hat{x} = b + \delta b. \tag{11}$$

Here A denotes the original matrix, b gives the constant values, and $\hat{x}$ gives the solution calculated using the algorithm. F represents the adjustments to A, and δb the adjustment to b that provides a problem that would result in the calculated solution using exact arithmetic. In this analysis the simplest case is given, namely where the algorithm uses $M = 1$.

In the first stage of condensation, $A^{(2)}$ is computed from $A^{(1)}$, which is the original matrix. It should be noted that each condensation step also incurs error on the right-hand side due to the algorithm carrying those values along during reduction. This error must also be accounted for in each step, so in the first stage of condensation, $b^{(2)}$ is computed from $b^{(1)}$ just as A.

Please cite this article in press as: K. Habgood, I. Arel, A condensation-based application of Cramer's rule for solving large-scale linear systems, Journal of Discrete Algorithms (2011), doi:10.1016/j.jda.2011.06.007

Before Chio's pivotal condensation occurs, the largest determinant is moved into the lead position. Since $M = 1$, the determinant is simply the value of the element. This greatly simplifies the conceptual nature for conveying this analysis. Normally $M \times M$ determinants would need to be calculated, and then all the rows comprising the largest determinant moved into the lead. Here, the row with the largest lead value a_{i1} is moved to row one, followed by each element in column one being divided by a_{kk}. This normalizes the matrix so that the absolute values of all row leads are smaller or equal to one, such that

$$a_{kj} = \frac{a_{kj}}{a_{kk}}(1 + \eta_{kj}), \quad \text{where } \eta_{kj} \leqslant \beta^{-t+1}. \tag{12}$$

In this case, β^{-t+1} is the base number system used for calculation with t digits. This is equivalent to machine epsilon, ϵ. The computed elements $a_{ij}^{(2)}$ are derived from this basic equation $a_{ij}^{(2)} = a_{ij} - a_{ik} \times a_{kj}$. When the error analysis is then added, we get:

$$a_{ij}^{(2)} = \left[a_{ij}^{(1)} - a_{ik} \times a_{kj} \times \left(1 + \gamma_{ij}^{(1)}\right)\right]\left(1 + \alpha_{ij}^{(1)}\right), \qquad \left|\gamma_{ij}^{(1)}\right| \leqslant \beta^{-t+1} \quad \text{and} \quad \left|\alpha_{ij}^{(1)}\right| \leqslant \beta^{-t+1}, \tag{13}$$

$$a_{ij}^{(2)} = a_{ij}^{(1)} - a_{ik} \times a_{kj} + e_{ij}^{(1)} \tag{14}$$

where

$$e_{ij}^{(1)} = \frac{a_{ij}^{(2)} \times \alpha_{ij}^{(1)}}{(1 + \alpha_{ij}^{(1)})} - a_{ik} \times a_{kj} \times \gamma_{ij}^{(1)}, \quad ij = 2, \ldots, n. \tag{15}$$

This then provides the elements for $E^{(1)}$ such that $A + E^{(1)}$ provides the matrix that would condense to $A^{(2)}$ with exact arithmetic. The lead column is given by $e_{ij}^{(1)} = a_{ik} \times \eta_{kj}$. This follows for each condensation step $A^{(k+1)} = A^{(k)} + E^{(k)}$ and similarly for the right-hand side, $b^{(k+1)} = b^{(k)} + E_b^{(k)}$, where E includes an additional column to capture the error incurred on b. In this case, the E matrix will capture the variability represented by δb found in Eq. (11). If taken through all steps of condensation, then $E = E^{(1)} + \cdots + E^{(n-1)}$, giving

$$(A + E)\hat{x} = b. \tag{16}$$

Bounds on E need evaluation, since this controls how different the matrix used for computation is from the original matrix. It's important to note that, due to the use of Cramer's rule, the algorithm can normalize a matrix and simply discard the value used to normalize. Cramer's rule is a ratio of values so as long as both values are divided by the same number the magnitude of that number is unimportant. This is a crucial trait, since needing to retain and later use these values would cause instability.

Consider $a = \max|a_{ij}|$, $g = \frac{1}{a}\max|a_{ij}^{(k)}|$ and Eq. (15). If these are combined along with the knowledge that $|a_{1j}| \leqslant 1$, the following is obtained

$$\left|e_{ij}^{(k)}\right| \leqslant \frac{\beta^{-t+1}}{1 - \beta^{-t+1}}\left|a_{ij}^{(k+1)}\right| + \beta^{-t+1} \times \left|a_{ij}^{(k)}\right| \leqslant \frac{2}{1 - \beta^{-t+1}} ag\beta^{-t+1}. \tag{17}$$

In essence the E matrix yields the following

$$|E| \leqslant ag\,\Upsilon \left\{ \begin{bmatrix} 0 & 0 & \ldots & \ldots & 0 \\ \beta^{-t+1} & 2 & \ldots & \ldots & 2 \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ \beta^{-t+1} & 2 & \ldots & \ldots & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & \ldots & 0 \\ 0 & 0 & 0 & \ldots & 0 \\ 0 & \beta^{-t+1} & 2 & \ldots & 2 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & \beta^{-t+1} & 2 & \ldots & 2 \end{bmatrix} + \cdots \right\}, \tag{18}$$

where $\Upsilon = \frac{\beta^{-t+1}}{(1-\beta^{-t+1})}$, such that

$$|E| \leqslant ag\,\Upsilon \begin{bmatrix} 0 & 0 & \ldots & \ldots & 0 \\ \beta^{-t+1} & 2 & \ldots & \ldots & 2 \\ \vdots & 2 + \beta^{-t+1} & 4 & \ldots & 4 \\ \vdots & \vdots & \vdots & & \vdots \\ \beta^{-t+1} & 2 + \beta^{-t+1} & 4 + \beta^{-t+1} & \ldots & 2n \end{bmatrix}. \tag{19}$$

The bottom row of the matrix clearly provides the largest possible value, whereby the summation is roughly n^2. When combined with the other factors, it yields the equality $\|E_\infty\| \leqslant 2n^2 ag\frac{\beta^{-t+1}}{1-\beta^{-t+1}}$. If it's assumed that $1 - \beta^{-t+1} \approx 1$ and a is simply a scaling factor of the matrix, two values of interest are left: n^2 and the growth factor g. The growth factor is the element that has the greatest impact on the overall value, since it provides a measure of the increase in value

Please cite this article in press as: K. Habgood, I. Arel, A condensation-based application of Cramer's rule for solving large-scale linear systems, Journal of Discrete Algorithms (2011), doi:10.1016/j.jda.2011.06.007

Table 1
Relative residual measurements for Cramer's algorithm.

Matrix size	$n\epsilon_{machine}$	Cramer's $\frac{\|b-A\hat{x}\|}{\|A\|\cdot\|\hat{x}\|}$	Matlab $\frac{\|b-A\hat{x}\|}{\|A\|\cdot\|\hat{x}\|}$
1000×1000	2.22E−13	5.93E−14	8.05E−16
2000×2000	4.44E−13	5.42E−14	1.04E−15
3000×3000	6.66E−13	9.62E−14	1.95E−15
4000×4000	8.88E−13	3.32E−13	2.49E−15
5000×5000	1.11E−12	8.12E−14	3.05E−15
6000×6000	1.33E−12	5.52E−14	3.35E−15
7000×7000	1.55E−12	7.46E−14	3.55E−15
8000×8000	1.78E−12	8.12E−14	4.28E−15

over numerous condensation steps. Fortunately, this value is bound because all multipliers are due to the pivoting and the division performed before each condensation, such that

$$\max\left|a_{ij}^{(k+1)}\right| = \max\left|a_{ij}^{(k)} - a_{ik} \times a_{kj}\right| \leqslant 2 \times \max\left|a_{ij}^{(k)}\right|. \tag{20}$$

The value of $a_{ij}^{(k)}$ is at most the largest value in the matrix. The value of $a_{ik} \times a_{kj}$ is also at most the largest value in the matrix. Since a_{ik} is the row lead, it's guaranteed to be one or less. The value of a_{kj} could possibly be the largest value in the matrix. Therefore, the greatest value that could result from the equation $a_{ij}^{(k)} - a_{ik} \times a_{kj}$ is twice the maximum value in the matrix or $2\max|a_{ij}^{(k)}|$. This can then repeat at most n times, which results in a growth factor of 2^n. The growth factor given for LU-decomposition with partial pivoting in the literature is $g \leqslant 2^{n-1}$ [13]. The slight difference being that this algorithm computes a solution directly, whereas LU-decomposition analysis must still employ forward and backward substitution to compute a solution vector. As with LU-decomposition, it can be seen that generally the growth rate will less than double at each step. In fact, the values tend to cancel each other leaving the growth rate around 1 in actual usage.

Mirroring does not affect the stability analysis of the algorithm. The matrices that are used to calculate the answers may have been mirrored numerous times. Since no calculations take place during the mirroring, and it does not introduce an additional condensation step, the mirroring has no bearing on the accuracy.

3.2. Backward error measurements

One of the most beneficial traits of LU-decomposition is that, although it has a growth rate of 2^{n-1}, in practice it generally remains stable. This can be demonstrated by relating the relative residual to the relative change in the matrix, giving the following inequality:

$$\frac{\|b - A\hat{x}\|}{\|A\| \cdot \|\hat{x}\|} \leqslant \frac{\|E\|}{\|A\|}. \tag{21}$$

The symbol $\|\hat{x}\|$ represents the norm of the calculated solution vector. When the residual found from this solution set is divided by the norm of the original matrix multiplied by the norm of the solution set, an estimate is produced of how close the solved problem is to the original problem. If an algorithm produces a solution to a problem that is very close to the original problem then the algorithm is considered stable. A reasonable expectation for how close the solved and given problems should be is expressed as [9].

$$\frac{\|E\|}{\|A\|} \approx n\epsilon_{machine}. \tag{22}$$

A pragmatic value of $\epsilon_{machine} \approx 2.2 \times 10^{-16}$ reflects the smallest value the hardware can accurately support, and n represents the size of the linear system. Table 1 shows this relative residual calculations when using Cramer's rule in comparison to those obtained with Matlab and for the target values given by Eq. (22). The infinite norm is used for all norm calculations and Cramer's rule used a condensation step size (M) of 8. As shown in Table 1, both Matlab's implementation of LU-decomposition and Cramer's rule deliver results below the target level to suggest a stable algorithm for the test matrices considered. The latter were created by populating the matrices with random values between -5 and 5 using the standard C language random number generator. Results produced by the proposed algorithm for these matrices were measured over numerous trials.

Matlab includes a number of test matrices in a matrix gallery that were used for further comparison. In particular, a set of dense matrices from this gallery were selected. Each type had four samples of 1000×1000 matrices. In many cases, Cramer's rule resorted to a condensation size of one ($M = 1$) for improved accuracy. The relative residuals were then calculated in the manner discussed above. Table 2 provides a summary of those findings. The results show relative residuals near Matlab for a majority of the specialized matrices.

Please cite this article in press as: K. Habgood, I. Arel, A condensation-based application of Cramer's rule for solving large-scale linear systems, Journal of Discrete Algorithms (2011), doi:10.1016/j.jda.2011.06.007

Table 2
Comparisons using Matlab matrix gallery.

Matrix type	Cramer's $\frac{\|b-A\hat{x}\|}{\|A\|\cdot\|\hat{x}\|}$	Matlab $\frac{\|b-A\hat{x}\|}{\|A\|\cdot\|\hat{x}\|}$
chebspec – Chebyshev spectral differentiation matrix	1.95E−07	1.13E−16
clement – Tridiagonal matrix with zero diagonal entries	3.66E−16	2.80E−17
lehmer – Symmetric positive definite matrix	2.67E−15	6.46E−18
circul – Circulant matrix	6.39E−14	8.35E−16
lesp – Tridiagonal matrix with real, sensitive eigenvalues	1.22E−16	1.43E−18
minij – Symmetric positive definite matrix	2.44E−15	2.99E−18
orthog – Orthogonal and nearly orthogonal matrices	1.41E−17	6.56E−17
randjorth – Random J-orthogonal matrix	1.40E−09	7.24E−16
frank – Matrix with ill-conditioned eigenvalues	5.59E−03	1.52E−21

Table 3
Algorithm relative error when compared to Matlab and GSL solution sets.

Matrix size	$\kappa(A)$	Matlab $\|x-\hat{x}\|_\infty$	GSL $\|x-\hat{x}\|_\infty$	Avg Matlab	Avg GSL
1000 × 1000	506 930	2.39E−9	1.93E−10	1.03E−10	5.38E−12
2000 × 2000	790 345	4.52E−9	5.36E−9	1.01E−10	7.27E−12
3000 × 3000	1 540 152	1.95E−8	1.84E−8	1.12E−10	2.09E−11
4000 × 4000	12 760 599	4.81E−8	5.62E−8	1.43E−10	7.91E−11
5000 × 5000	765 786	2.92E−8	4.39E−8	1.18E−10	3.46E−11
6000 × 6000	1 499 430	8.67E−8	8.70E−8	1.37E−10	6.04E−11
7000 × 7000	3 488 010	9.92E−8	8.95E−8	1.27E−10	5.15E−11
8000 × 8000	8 154 020	9.09E−8	9.43E−8	1.86E−10	7.85E−11

Table 4
Execution time comparison to LAPACK.

Matrix size	Algorithm (s)	Matlab (s)	Ratio
1000 × 1000	2.06	.91	2.26
2000 × 2000	16.44	6.32	2.60
3000 × 3000	52.33	19.92	2.63
4000 × 4000	115.44	45.10	2.56
5000 × 5000	220.32	86.90	2.54
6000 × 6000	380.92	142.05	2.68
7000 × 7000	583.02	242.61	2.40
8000 × 8000	872.26	334.68	2.61

3.3. Forward measurements

The forward error is typically defined as the relative difference between the true values and the calculated ones. Here, the actual answers are generated by Matlab and GSL (GNU scientific library). The solution vector provided by the algorithm was compared to those given by both software packages. Table 3 details the observed relative difference between the software packages and the solutions provided by the proposed algorithm.

4. Implementation results

The algorithm has been implemented on a single processor platform. Compiled in the C programming environment, it has been compared to LAPACK (Linear Algebra PACKage) via Matlab on a single core Intel Pentium machine, to provide a baseline for comparison. The processor was an Intel Pentium M Banias with a frequency of 1.5 GHz using a 32 KB L1 data cache and 1 MB L2 cache. The manufacturer quotes a maximum GFLOPS rate of 2.25 for the specific processor [7]. The Linpack benchmark MFLOPS for this processor is given as 755.35.

Several optimization techniques have been applied to the implementation, including the SIMD (single instruction multiple data) parallelism that is standard on most modern processors. The program code also employs memory optimizations such as cache blocking to reduce misses. No multi-processor parallelization has been programmed into the implementation such that the algorithm itself could be evaluated against the industry standard prior to any parallelization efforts. The implementation has checks to ensure a certain levels of accuracy. In certain matrices, as mentioned in Section 3.2, the algorithm may drop to a lower execution speed to ensure accuracy of the results.

As can be seen in Table 4, the algorithm runs approximately 2.5 times slower than the execution time of Matlab, independent of matrix size, which closely corresponds to the theoretical complexity analysis presented above. Both pieces of software processed numerous trials on a 1.5 GHz single core processor. The results further show that while the algorithm is slower than the LU-based technique, it is consistent. Even as the matrix sizes grow, the algorithm remains roughly 2.5 times slower than state of the art methodologies. Fig. 2 depicts a comparison between the proposed algorithm and Matlab.

Please cite this article in press as: K. Habgood, I. Arel, A condensation-based application of Cramer's rule for solving large-scale linear systems, Journal of Discrete Algorithms (2011), doi:10.1016/j.jda.2011.06.007

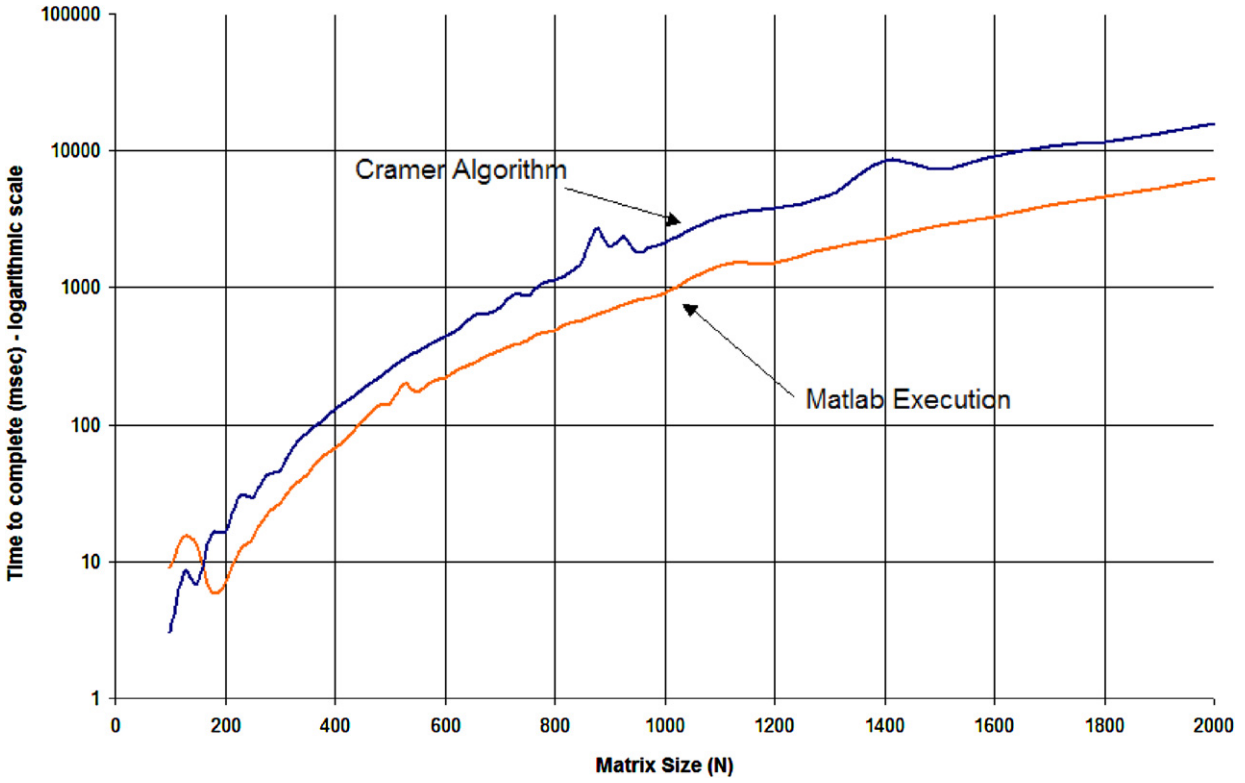

Fig. 2. Algorithm execution times compared to those obtained using Matlab(TM).

The theoretical number of floating point operations (FLOPS) to complete a 1000×1000 matrix based on the complexity calculation is roughly 1555 million. The actual measured floating point operations for the algorithm summed to 1562.466 million. This equates to an estimated 758 MFLOPS. The Matlab algorithm measured 733 MFLOPS based on the measured execution time and theoretical number of operations for LU decomposition.

5. Conclusions

To the best of our knowledge, this is the first paper outlining how Cramer's rule can be applied in a scalable manner. It introduces a novel methodology for solving large-scale linear systems. Unique utilization of Cramer's rule and matrix condensation techniques yield an elegant process that has promise for parallel computing architectures. Implementation results support the theoretical claims that the accuracy and computational complexity of the proposed algorithm are similar to LU-decomposition.

Acknowledgements

The authors would like to thank Prof. Jack Dongarra and the Innovative Computing Laboratory (ICL) at the University of Tennessee for their insightful suggestions and feedback.

References

[1] M. Abramowitz, I.A. Stegun, Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, 9th printing, Dover, 1972, p. 10.
[2] A.C. Aitken, Determinants and Matrices, sixth ed., Oliver and Boyd, 1956.
[3] S.C. Chapra, R. Canale, Numerical Methods for Engineers, second ed., McGraw-Hill Inc., 1998.
[4] S.A. Dianant, E.S. Saber, Advanced Linear Algebra for Engineers with Matlab, CRC Press, 2009, pp. 70–71.
[5] C. Dunham, Cramer's rule reconsidered or equilibration desirable, ACM SIGNUM Newsletter 15 (1980) 9.
[6] H.W. Eves, Elementary Matrix Theory, Courier Dover Publications, 1980.
[7] Intel_Corporation, Intel microprocessor export compliance metrics, August 2007.
[8] N. Galoppo, N.K. Govindaraju, M. Henson, D. Manocha, LU-GPU: Efficient algorithms for solving dense linear systems on graphics hardware, in: SC, IEEE Computer Society, 2005, p. 3.
[9] M.T. Heath, Scientific Computing: An Introductory Survey, McGraw-Hill, 1997.
[10] N.J. Higham, Accuracy and Stability of Numerical Algorithms, Society for Industrial and Applied Mathematics, SIAM, 1996.

Please cite this article in press as: K. Habgood, I. Arel, A condensation-based application of Cramer's rule for solving large-scale linear systems, Journal of Discrete Algorithms (2011), doi:10.1016/j.jda.2011.06.007

[11] G.E. Karniadakis, R.M. Kirby II, Parallel Scientific Computing in C++ and MPI: A Seamless Approach to Parallel Algorithms and Their Implementation, Cambridge University Press, 2003.
[12] C. Moler, Cramer's rule on 2-by-2 systems, ACM SIGNUM Newsletter 9 (1974) 13–14.
[13] J.M. Ortega, Numerical Analysis: A Second Course, Academic Press, 1972.

Please cite this article in press as: K. Habgood, I. Arel, A condensation-based application of Cramer's rule for solving large-scale linear systems, Journal of Discrete Algorithms (2011), doi:10.1016/j.jda.2011.06.007