Input randomization and automatic derivation of systolic arrays for matrix computations

Dinh Lê D. Stott Parker Brad Pierce

Computer Science Department, UCLA School of Engineering and Applied Science
Los Angeles, CA 90095-1596, USA

Abstract

Many standard matrix algorithms are difficult to implement as systolic arrays, because they involve data movement that cannot be determined a priori, resulting in high timing complexity. In the case of dense matrix inversion, Gaussian elimination (GE) with pivoting has largely been supplanted by alternative schemes, such as Givens rotations and the Gram-Schmidt method, which are costly and complex, and the simpler GE with pairwise pivoting and GE without pivoting, which can break down on "degenerate" inputs, *i.e.*, invertible matrices with noninvertible submatrices.

A new input randomization technique efficiently transforms the linear problem $A\mathbf{x} = \mathbf{b}$ into the randomized problem $(VA)\mathbf{x} = (V\mathbf{b})$, where the matrix V is chosen from a special class of random matrices. If A is nonsingular, then, with probability 1, GE without pivoting can be successfully applied to VA. As is demonstrated by an extended tutorial example, this simple algorithm is amenable to current-generation matrix algorithm compilers, such as UCLA's Mamacg system.

1 Introduction

Systolic architectures were popularized by Kung [2]. Many techniques have been developed since to exploit their regularity and potential for massive parallelism.

The heart of a systolic system is a regular network of processors such as the 4×4 mesh array shown in Figure 1. Input arrives (from the north and west), pulses in a wavefront ('systolic') manner through the system, and exits (from the south and east). In Figure 1, two 4×4 input matrices, A and B, arrive at the west and north ends, respectively. Seven pulses are required to send and process data from processor P_{11} to processor P_{44}. The two 4×4 output matrices, C and D, emerge from the east and south. Although our interest is restricted to mesh arrays, it should be noted that various other topologies have been discussed in the literature, including the pipeline, triangle, hexagon and helix.

1.1 Systolic Gaussian Elimination and Pivoting

Gaussian elimination with pivoting is a classic example of a matrix algorithm that is difficult to implement as a systolic algorithm. In pivoting, the maximum entry in the current column is computed, then, based on the location of this element, two rows in the current matrix are swapped. This entails data movement that cannot be determined a priori and amounts to a dynamically constructed permutation of the original problem. The overhead of pivoting is high: Megson [4, p.199] notes that "These new designs are

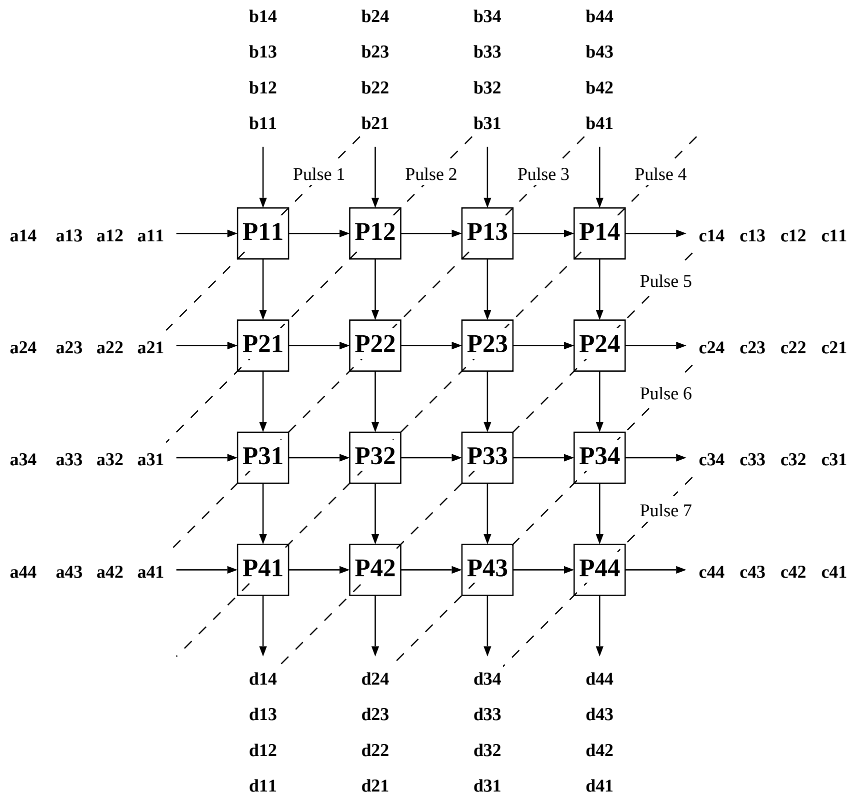


Figure 1: A 4×4 systolic architecture

based on extensions to the triangular arrays of Gentleman and Kung [1] and require $O(n^2)$ time and $O(n^2)$ cells to implement partial pivoting. This is in contrast to the nonpivoting and nearest neighbor pivoting schemes which require $O(n)$ computation time and $O(n^2)$ cells." Among nonpivoting and pairwise pivoting systems are ordinary GE (LU decomposition without pivoting), Givens rotations, and the Gram-Schmidt method. While Givens rotations and Gram-Schmidt require costly and complex computations, ordinary GE without pivoting and matrix triangularization with pairwise pivoting[1] [1] fail to handle a common kind of *degeneracy*: an invertible input matrix with a leading principal submatrix that is not invertible.

In this paper, we survey our recent results [8, 9] showing that input randomization can be a general tool for regularizing array computations, and demonstrate that this simplification makes practical the semi-automated construction of systolic arrays (and other regular parallel architectures) for important applications in numerical linear algebra. After explaining our input randomization technique, we give an extended tutuorial example of the derivation, using UCLA's MAMACG software [3], of a systolic rank-4 GE

[1]The pairwise pivoting scheme was used in [1] mainly to avoid global communication and to improve numerical stability.

architecture for solving systems of linear equations.

1.2 Related Literature

There is an extensive literature [10] on the design of systolic arrays for the direct solution of dense systems of linear equations, and, more generally, for the solution of the algebraic path problem. Megson's *Introduction to Systolic Algorithm Design* (1992) [6] is a good survey, emphasizing application and performance.

An important subliterature has grown up around so-called *Faddeev arrays*, which compute Schur complements. Given a rectangular matrix $M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$ with an invertible leading submatrix A, the Schur complement of A with respect to M, denoted $(\ M\ /\ A\ )$, is defined as $D - CA^{-1}B$. By using suitable inputs, a Faddeev array can be flexibly used for a variety of computations, *e.g.*, setting $D = 0$ and $C = -I$ yields $A^{-1}B$. Although Faddeev arrays are often said to implement "Faddeev's algorithm", according to Megson [5, p. 1594] this algorithm is "just a special case of dense matrix triangularization without pivoting". Perhaps the confusion stems from the fact that in Faddeeva's classic *Computational Methods of Linear Algebra* (1959) she discusses both Schur complements (§14) and the Souriau-Frame-Faddeev algorithm (§25). Megson [5, p. 1594] contrasts the Schur complement and algebraic path problem approaches:

> "Faddeev's algorithm is ... of interest because of its generality in solving a range of linear algebra problems (e.g., solution for multiple right-hand sides, matrix-matrix, matrix-vector products, Schur's complement of a matrix, and matrix inversion). Indeed the generality is obtained by a careful choice of the input data rather than exploiting the algebraic structures attributed to semi-rings previously used in the literature to produce generic arrays based on switching of operations at the cell level ..."

The Souriau-Frame-Faddeev algorithm[2] from linear control theory has received little attention from the systolic array community. There has been some disagreement on the practicality of the algorithm, since it is expensive and can be numerically unstable. The algorithm is simple, however, and handles all matrices except those lacking distinct eigenvalues. We suspect, but have not yet proved, that our randomization scheme can also remove this form of degeneracy.

2 A Randomization Technique

The randomization technique was pioneered in [8]. The particular variant we describe here is developed in [9]. Let us sketch the idea of this randomizing linear transformation, and how it can be used.

2.1 The Basic Idea

Let A be a nonsingular $n \times n$ matrix and V be a random $n \times n$ matrix. Our basic idea is that, given the randomness of V, *we can perform Gaussian elimination on the*

[2]Discovered independently by J.M. Souriau, J.S. Frame, and D.K. Faddeev in the late 1940s, this algorithm is sometimes known as the "Leverrier-Faddeev" algorithm or simply as "Faddeev's" algorithm.

product matrix VA without any pivoting. Intuitively VA is sufficiently 'random' that, with probability 1, pivoting is never needed.

Specifically, suppose A is a nonsingular real $n \times n$ matrix, let V be a nonsingular random $n \times n$ matrix. By a random matrix, we mean a matrix whose elements are chosen independently from a suitable distribution. We can guarantee such matrices to be nonsingular by choosing them from a nonsingular class of matrices. Then:

- To solve $A\mathbf{x} = \mathbf{b}$, we can instead solve $(VA)\mathbf{x} = V\mathbf{b}$.
- To compute A^{-1}, we can instead compute $(VA)^{-1}$, since VA is guaranteed to be nonsingular. Afterwards, it follows that $A^{-1} = (VA)^{-1}V$.
- To compute $det\, A$, we can instead compute $det\,(VA)/(det\, V)$, assuming the determinants of V can be computed easily.

Each of the computations here is easily implemented if Gaussian elimination without pivoting successfully factors the preconditioned matrix VA.

Putting it another way, Gaussian elimination needs pivoting to handle **degenerate matrices** — matrices that have a singular leading principal submatrix. On these matrices elimination will produce either a zero on the diagonal and then division by zero or, because of roundoff error, extreme cancellation on the diagonal and then division by inaccurate values. For nonsingular matrices this degeneracy is an artifact of the vector basis used, and is not an *invariant* property of the matrix. Thus we can transform any problem involving a possibly degenerate matrix A into a problem involving a (randomized, almost certainly) nondegenerate matrix VA.

2.2 Example: Using Randomization instead of Pivoting

To illustrate the basic idea, consider the 2×2 matrix

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

If we arbitrarily select the random matrix

$$V = \begin{pmatrix} +.6483 & +.7614 \\ +.6483 & -.7614 \end{pmatrix}$$

we get the random transform

$$VA \quad = \quad \begin{pmatrix} +.6483 & +.7614 \\ +.6483 & -.7614 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad = \quad \begin{pmatrix} .7614 & .6483 \\ -.7614 & .6483 \end{pmatrix}.$$

Gaussian elimination can be applied directly to this matrix without any use of pivoting. For example, with $\mathbf{b} = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$,

$$V\,\mathbf{b} \quad = \quad \begin{pmatrix} +.6483 & +.7614 \\ +.6483 & -.7614 \end{pmatrix} \begin{pmatrix} 2 \\ 3 \end{pmatrix} \quad = \quad \begin{pmatrix} 3.581 \\ -.987 \end{pmatrix}.$$

Ordinary Gaussian elimination without pivoting on $V\,A\,\mathbf{x} = V\,\mathbf{b}$ then gives

$$\mathbf{x} \quad = \quad \begin{pmatrix} 2.999 \\ 2.001 \end{pmatrix}$$

which is correct to three digits.

2.3 What Randomization Must Do to Replace Pivoting

Gaussian elimination is a sequence of transformations to an $n \times n$ matrix $A = (a_{ij})$, reducing it to upper-diagonal form in $n-1$ steps. With the initial assignment $a_{ij}^{(1)} = a_{ij}$, it can be equationally defined for $1 \leq k \leq n-1$ as

$$a_{ij}^{(k+1)} = \begin{cases} 0 & i \geq k+1,\ j = k \\ a_{ij}^{(k)} - \frac{a_{ik}^{(k)} a_{kj}^{(k)}}{a_{kk}^{(k)}} & i \geq k+1,\ j \geq k+1 \\ a_{ij}^{(k)} & \text{otherwise.} \end{cases}$$

Gaussian elimination works only when, for all $1 \leq k \leq n-1$, the $a_{kk}^{(k)}$ are nonzero. It is well-known that this is equivalent to the following:

> An $n \times n$ matrix A is **Gauss eliminable** if all its leading principal submatrices are nonsingular.

As above, assume that the process of randomization transforms a nonsingular input matrix A to VA, for some random matrix V. To be useful in Gaussian elimination, then, this randomizing transform needs to accomplish several things:

- The result of randomization must be Gauss eliminable.
- The cost of randomization must be reasonable.
- The roundoff errors incurred in the randomization, and in the subsequent Gaussian elimination (without pivoting), must be acceptable.

Simply multiplying by an arbitrary random matrix will achieve the first goal. However, this multiplication will be expensive, requiring more time than Gaussian Elimination itself. Furthermore, multiplying by poorly-conditioned random matrices can cause serious loss of numerical accuracy.

2.4 The Random DFT and FFT

In [9] we describe a specific randomizing transform that is both efficient and has the property that, for any nonsingular matrix A, the transform of A is Gauss eliminable with probability 1. Thus, with extremely high probability, Gaussian elimination will succeed.

The **Discrete Fourier Transform (DFT)** matrix[3] is

$$F_n \quad = \quad \frac{1}{\sqrt{n}} \left({\omega_n}^{(i-1)(j-1)} \right),$$

where $\omega_n = exp(-\frac{2\pi}{n}\,\mathsf{i})$. When n is a power of 2, the **Fast Fourier Transform (FFT)** results from factoring F_n into recursive products of butterfly transformations and permutations.

A **Random Discrete Fourier Transformation (RDFT)** of a square $n \times n$ matrix A is a product

$$\widetilde{A} \quad = \quad F_n\ D\ A\,,$$

where D is a nonsingular random $n \times n$ diagonal matrix.

[3] We index from 1 to n.

For example, note that when $n = 2$ if we have

$$F_2 = \begin{pmatrix} +.7071 & +.7071 \\ +.7071 & -.7071 \end{pmatrix}, \qquad D = \begin{pmatrix} 1.078 & 0 \\ 0 & .9168 \end{pmatrix}$$

then the matrix V in the example above is recognizable as $V = F_2\, D$.

The RDFT can be used to implement Gaussian Elimination without pivoting:

Theorem

The RDFT yields Gauss eliminable matrices. That is, with probability 1, Gaussian elimination can operate without any pivoting on $\widetilde{A} = VA$, the result of the RDFT.

The proof, presented in [9], shows that leading principal submatrices of $\widetilde{A}$ are almost always nonsingular. It does this by noting that the determinants of these matrices are all polynomials in the independent random variables (diagonal entries) of D, and that these polynomials are nonzero with probability 1. (Block submatrices of F_n are multiples of Vandermonde matrices, with nonzero determinant, and the Binet-Cauchy theorem for determinants gives explicit formulas for the polynomials, which can then be shown to be nonzero as stated.) Thus $\widetilde{A}$ is Gauss eliminable.

In fact, this Theorem holds for arbitrary $n > 0$, and not just for powers of 2. So we need not specifically rely on the FFT, but can use any implementation of the DFT F_n.

The RDFT has the drawback that it requires complex arithmetic for $n > 2$. In [8] a related transform, called the Random Butterfly Transform (RBT), is described that has essentially the same computational graph as the RFFT, but enables the use of real arithmetic for real inputs.[4] We have used the FFT here since most readers are probably familiar with it, and it has many systolic implementations.

2.5 Roundoff Error with Randomization

The magnitude of roundoff errors incurred without pivoting can be very significant when the matrix A is poorly conditioned. In [8] we discuss roundoff errors resulting under randomization. Although it is possible to derive bounds for the backwards error in terms of determinants of matrix elements, the randomized version of Gaussian elimination is less stable than that with pivoting. It is possible that pairwise pivoting [1] would be ameliorative in practice. However, ultimately a statistical error analysis like that in [11] is needed to test any variant of Gaussian elimination. Backwards error analyses give only partial certification of a numerical algorithm.

3 A Randomized Systolic Linear System Solver

Let A be an $n \times n$ matrix and $\mathbf{b}$ be a vector of size n. A randomized GE algorithm for solving the system of linear equations $A\mathbf{x} = \mathbf{b}$ can be implemented as:

```
procedure randomized.solve(n, A, b) =
1.  D = diag(rand(1, n))
2.  A = fft(D × A)
```

[4] Simply using the fast Hartley transform in place of the FFT does not work.

3. $\mathbf{b} = \text{fft}(D \times \mathbf{b})$
4. ordinary.GE$(n, A, \mathbf{b})$
5. back.substitution$(n, A, \mathbf{b})$

Statements 1, 2, and 3 comprise a randomization technique described in Section 2 where fft$(D \times A) = VA$. Statement 4 performs ordinary LU decomposition and back-substitution on the system of linear equations $(VA)\mathbf{x} = (V\mathbf{b})$, storing the result in the vector $\mathbf{b}$. A complete randomized systolic architecture for solving systems of linear equations, as shown in Figure 2, can be implemented in three units: randomization (statements 1–3), ordinary GE (statement 4), and back-substitution (statement 5).

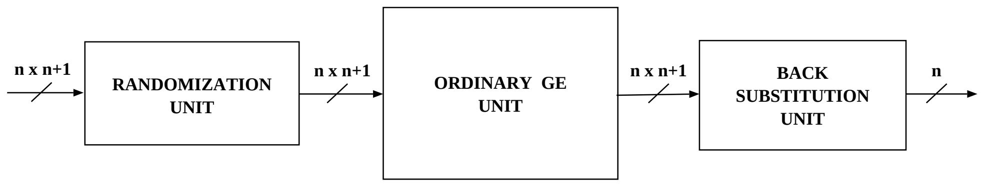


Figure 2: Architecture of the randomized systolic linear system solver

The randomization amounts to random row scaling and the FFT. Many systolic implementations have been proposed for the FFT and any one of these could be used here. Rather than specialize the discussion to a particular systolic implementation, we will rely on a FFT computational-graph. Consider the rank-4 case. Using the FFT decomposition of the DFT matrix

$$F_4 \quad = \quad \frac{1}{\sqrt{4}} \begin{pmatrix} {\omega_4}^0 & {\omega_4}^0 & {\omega_4}^0 & {\omega_4}^0 \\ {\omega_4}^0 & {\omega_4}^1 & {\omega_4}^2 & {\omega_4}^3 \\ {\omega_4}^0 & {\omega_4}^2 & {\omega_4}^4 & {\omega_4}^6 \\ {\omega_4}^0 & {\omega_4}^3 & {\omega_4}^6 & {\omega_4}^9 \end{pmatrix} \quad = \quad \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -\mathrm{i} & -1 & \mathrm{i} \\ 1 & -1 & 1 & -1 \\ 1 & \mathrm{i} & -1 & -\mathrm{i} \end{pmatrix},$$

where $\omega_4 = \exp^{-2\pi\mathrm{i}/4} = -\mathrm{i}$, we obtain the factorization

$$V \quad = \quad F_4\, D \quad = \quad \frac{1}{2} \left(\begin{array}{cc|cc} 1 & & 1 & \\ & 1 & & -\mathrm{i} \\ \hline 1 & & -1 & \\ & 1 & & \mathrm{i} \end{array} \right) \left(\begin{array}{cc|cc} 1 & 1 & & \\ 1 & -1 & & \\ \hline & & 1 & 1 \\ & & 1 & -1 \end{array} \right) \left(\begin{array}{cc|cc} d_1 & & & \\ & & d_3 & \\ \hline & d_2 & & \\ & & & d_4 \end{array} \right)$$

where d_1, d_2, d_3, d_4 are the elements of the random diagonal matrix D. (Random values can be generated efficiently in silicon with, for example, feedback shift-registers.) Ignoring random number generation, the computational-graph of the rank-4 randomization unit can be drawn as in Figure 3.

We can now adapt the ordinary GE computational graph in Figure 4, which produces L, U and $\mathbf{y}$. This systolic computational graph can be mapped directly to a systolic system by treating each cell as a processor.

Since the upper triangular matrix U and the intermediate vector $\mathbf{y}$ resulting from ordinary GE are passed on to the back-substitution unit, the MMGs for randomization, ordinary GE, and back-substitution can be composed to form a MMG that solves systems of linear equations, as shown in Figure 5.

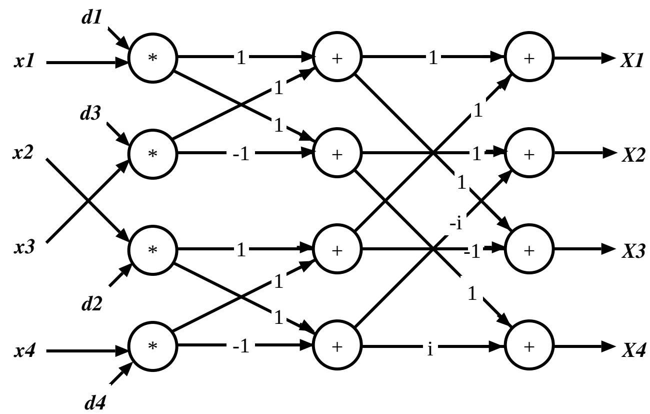


Figure 3: Computational-graph of a rank-4 randomization unit

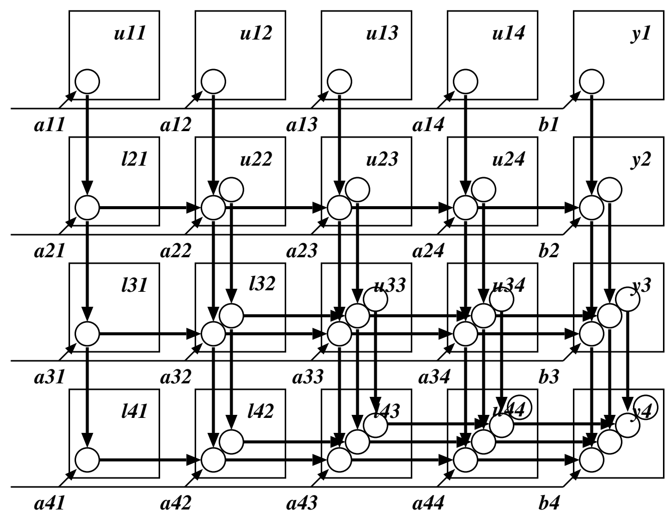


Figure 4: A systolic rank-4 ordinary GE computational-graph

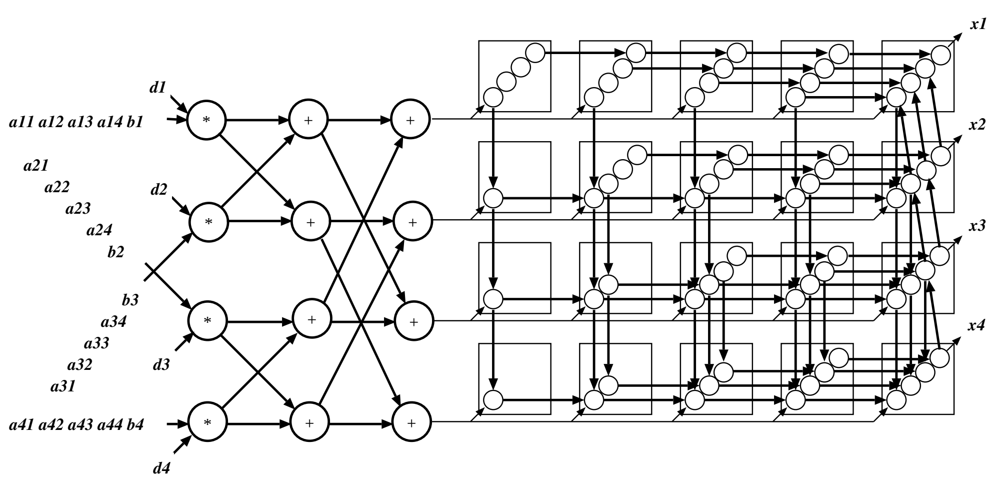


Figure 5: A systolic array solving rank-4 linear systems without pivoting

4 Derivation of Systolic Computational Graphs

Using MAMACG, a systolic system for solving a system of linear equations using the ordinary GE and back-substitution algorithms is derived in the following steps:

1. The algorithms are transformed to 3D-matrix format.
2. the resulting algorithms are symbolically executed ('unrolled', 'unfolded'), generating symbolic statements.
3. The symbolic statements are mapped to 3D orthogonal data-dependency graphs (ODGs).
4. The ODGs are transformed to 3D multimesh graphs (MMGs).
5. Prisms of nodes of size $p \times q$ are grouped ('folded') along axes of the MMG to obtain a 2D systolic computational graph for solving system of linear equations.

Figure 6 shows the steps outlined above. The mapping of matrix algorithms to their

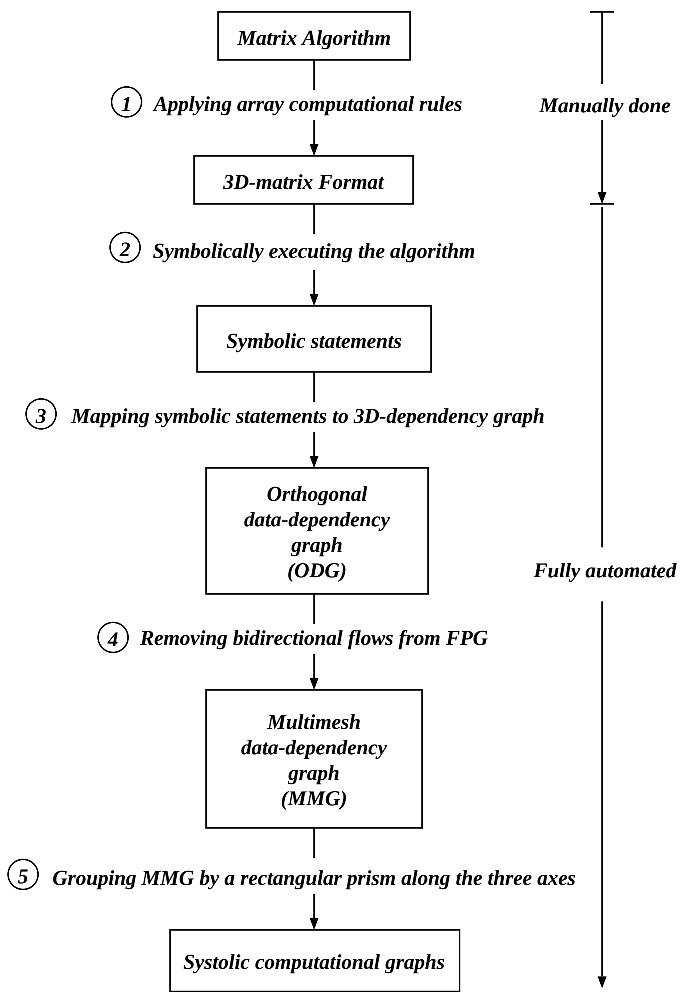


Figure 6: Steps taken to derive systolic computational graphs

3D-matrix formats currently requires user intervention, whereas the other steps are fully automated. However, the mapping is straightforward in many cases, as demonstrated in the next section.

4.1 Transforming Matrix Algorithms to 3D-Matrix Format

To be used by MAMACG, a matrix algorithm must first be transformed from 2D-matrix format to 3D-matrix format, using a procedure like the following, assuming input and output variables are specified as part of the algorithm:

1. At most three subscripted variables may appear on the right-hand-side expression of any assignment statement.
2. Each subscripted variable may appear on the left-hand-side expression only once (single assignment rule).
3. All 2-tuple subscripted variables must be expanded to 3-tuple subscripted variables while preserving the semantics of the algorithm. The added dimension reflects the temporal order of value assignments to the subscripted variables. With the loop

   for $k := 1$ **to** n **do** *BODY*

   all 2-tuple subscripted variables $\alpha[i, j]$ inside *BODY* can be expanded to $\alpha[i, j, k]$ (denoting $\alpha[i, j]$ at the k-th iteration) as follows:

   - A variable $\alpha[i, j]$ on the left-hand-side of a statement is expanded to $\alpha[i, j, k]$.
   - A variable $\alpha[i, j]$ on the right-hand-side of a statement is expanded to:
     - $\alpha[i, j, 0]$ if it is an input variable
     - $\alpha[i, j, k]$ if it is the left-hand-side of a previous *BODY* statement
     - $\alpha[i, j, k-1]$ otherwise.

4.2 Derivation of ODGs for GE and Back-Substitution

A system of linear equations $A\mathbf{x} = \mathbf{b}$, where $n \times n$ nonsingular matrix A and length n vector $\mathbf{b}$ are input, can be solved in two steps. First, ordinary GE is applied to the partitioned matrix $(A|\mathbf{b})$, and then back-substitution is used to solve the resulting triangular system for $\mathbf{x}$. Ordinary GE and back-substitution can be implemented as:

```
\\ Ordinary GE
\\ input: A_{n×(n+1)}
\\ output: L_{n×n}, U_{n×(n+1)}
var
  i, j, k: int;
begin
  for k := 1 to n do begin
    for j := k to n + 1 do
      U[k, j] := A[k, j];
    for i := k + 1 to n do
      L[i, k] := A[i, k] * U[k, k];
    for i := k + 1 to n do
      for j := k + 1 to n + 1 do
        A[i, j] := A[i, j] - L[i, k] * U[k, j];
  end
```

```
end

\\ Back-Substitution
\\ input: T_{n×(n+1)} (upper triangular)
\\ output: (n+1)-st column T[*, n+1]
var
   i, j, k: int;
begin
   for k := 1 to n do begin
      j := n - k + 1;
      T[j, n+1] := T[j, n+1]/T[j, j];
      for i := j - 1 downto 1 do
         T[i, n+1] := T[i, n+1] - T[i, j] * T[j, n+1];
   end
end
```

In the above code, the $(n+1)$-st column of matrix A holds the input vector $\mathbf{b}$ and the $(n+1)$-st column of matrix U holds the intermediate vector $\mathbf{y}$.

2D-Matrix Format

1.	$U[k,j]$	$=$	$A[k,j]$
2.	$L[i,k]$	$=$	$A[i,k] * U[k,k]$
3.	$A[i,j]$	$=$	$A[i,j] - L[i,k] * U[k,j]$
4.	$T[j,n+1]$	$=$	$T[j,n+1]/T[j,j]$
5.	$T[i,n+1]$	$=$	$T[i,n+1] - T[i,j] * T[j,n+1]$

$\downarrow$

3D-Matrix format

1.	$U[k,j,k]$	$=$	$A[k,j,k-1]$
2.	$L[i,k,k]$	$=$	$A[i,k,k-1] * U[k,k,k]$
3.	$A[i,j,k]$	$=$	$A[i,j,k-1] - L[i,k,k] * U[k,j,k]$
4.	$T[j,n+1,k]$	$=$	$T[j,n+1,k-1]/T[j,j,0]$
5.	$T[i,n+1,k]$	$=$	$T[i,n+1,k-1] - T[i,j,0] * T[j,n+1,k]$

The tables above show the transformations of the five statements appearing in the ordinary GE algorithm and back-substitution to their 3D-matrix format. All 2-tuple subscripted variables are expanded to 3-tuple subscripted variables with the added third indices reflecting their temporal orders.

The GE and back-substitution algorithms with the resulting 3D-matrix format can then be expanded (using loop unrolling) into a set of explicitly subscripted assignment statements. With the dependencies among these assignments, we obtain the desired ODGs (the 3D-orthogonal data-dependency graphs).

The derivation of the ODGs for rank-3 ordinary GE and back-substitution are shown in Figure 7 and Figure 8, respectively.

4.3 Transforming MMG to Systolic Computational Graphs

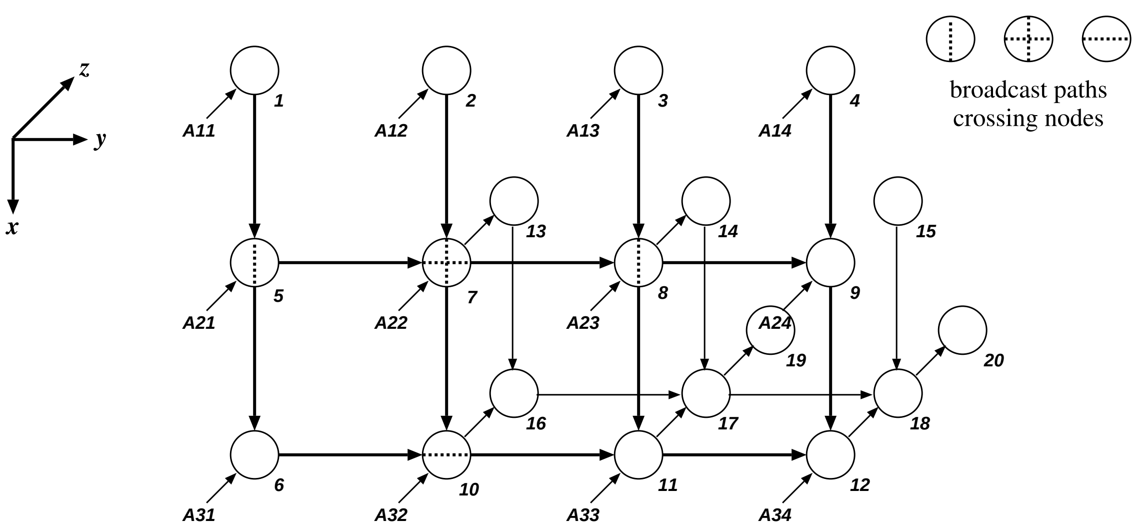
As the basis for efficient mesh array designs, the MMG method [7] introduces a 3D multimesh graph representation, the MMG. To obtain the MMG from the corresponding ODG, bidirectional flows must be removed. Since the ODGs for rank-3 ordinary GE and

```
    for k := 1 to 3 do
      begin
        for j := k to 4 do
1.          U[k,j,k] := A[k,j,k-1];
        for i := k+1 to 3 do
2.          L[i,k,k] := A[i,k,k-1]/U[k,k,k];
        for i := k+1 to 3 do
            for j := k+1 to 4 do
3.              A[i,j,k] := A[i,j,k-1] - L[i,k,k]*U[k,j,k];
      end
```

a) Ordinary GE code after instantiating n to 3

```
k = 1

(1)       U[1,1,1] := A[1,1,0]
(2)       U[1,2,1] := A[1,2,0]
(3)   1.  U[1,3,1] := A[1,3,0]
(4)       U[1,4,1] := A[1,3,0]

(5)   2.  L[2,1,1] := A[2,1,0] / U[1,1,1]
(6)       L[3,1,1] := A[3,1,0] / U[1,1,1]

(7)       A[2,2,1] := A[2,2,0] - L[2,1,1]*U[1,2,1]
(8)       A[2,3,1] := A[2,3,0] - L[2,1,1]*U[1,3,1]
(9)   3.  A[2,4,1] := A[2,1,0] - L[2,1,1]*U[1,4,1]
(10)      A[3,2,1] := A[3,2,0] - L[3,1,1]*U[1,2,1]
(11)      A[3,3,1] := A[3,3,0] - L[3,1,1]*U[1,3,1]
(12)      A[3,4,1] := A[3,4,0] - L[3,1,1]*U[1,4,1]

k = 2

(13)      U[2,2,2] := A[2,2,1]
(14)  1.  U[2,3,2] := A[2,3,1]
(15)      U[2,4,2] := A[2,4,1]

(16)  2.  L[3,2,2] := A[3,2,1] / U[2,2,2]

(17)  3.  A[3,3,2] := A[3,3,1] - L[3,2,2]*U[2,3,2]
(18)      A[3,4,2] := A[3,4,1] - L[3,2,2]*U[2,4,2]

k = 3

(19)  1.  U[3,3,3] := A[3,3,2]
(20)      U[3,3,3] := A[3,4,2]
```

b) Symbolic statements generated after 1st, 2nd and 3rd iterations



c) Orthogonal data-dependency graph

Figure 7: Deriving the ODG for rank-3 ordinary GE

```
for k := 1 to 3 do
  begin
    j := n-k+1
    T[j,4,k] := T[j,4,k-1]/T[j,j,0]
    for i := j-1 downto 1 do
      begin
        T[i,4,k] := T[i,4,k-1] - T[i,j,0]*T[j,4,k]
      end
  end
```

(a) Back substitution after instantiating n to 3

k=1

(1) T[3,4,1] := T[3,4,0]/T[3,3,0]
(2) T[2,4,1] := T[2,4,0] - T[2,3,0]*T[3,4,1]
(3) T[1,4,1] := T[1,4,0] - T[1,3,0]*T[3,4,1]

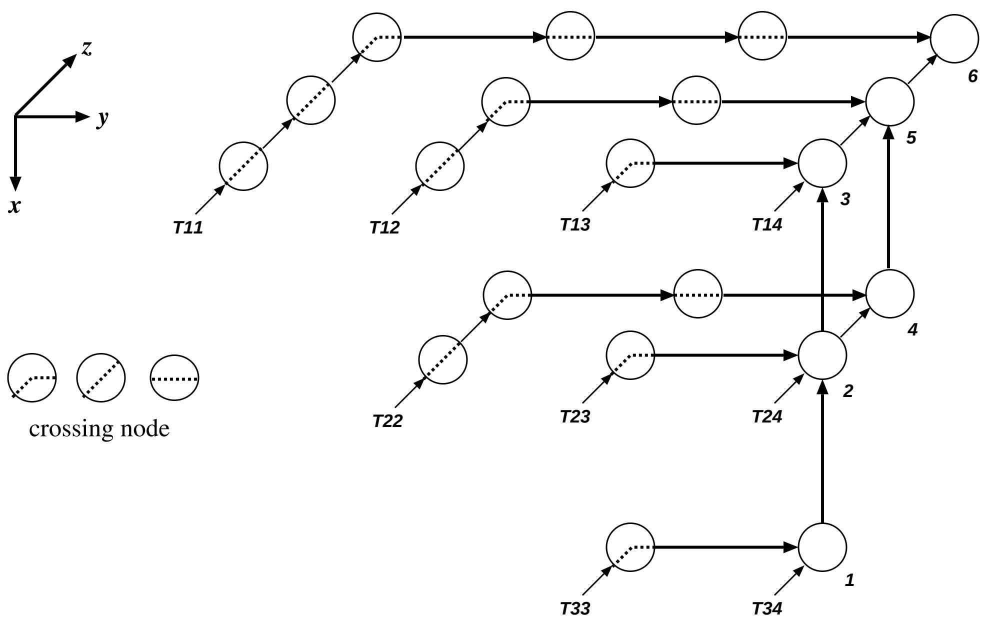
k=2

(4) T[2,4,2] := T[2,4,1]/T[2,2,0]
(5) T[1,4,2] := T[1,4,1] - T[1,2,0]*T[2,4,2]

k=3

(6) T[1,4,3] := T[1,4,2]/T[1,1,0]

(b) Symbolic statements generated after 1st, 2nd, and 3rd iterations



(c) Orthogonal Data-Dependency Graph

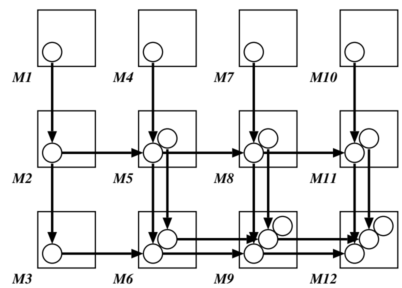
Figure 8: Deriving the ODG for rank-3 back-substitution

back-substitution do not contain bidirectional flow, their MMGs are their ODGs. ODGs of other systolic algorithms contain bidirectional flows, and these must be removed. Interested readers should consult [3] for details.

Since the resulting upper triangular matrix U and the intermediate vector $\mathbf{y}$ generated by ordinary GE is passed to back-substitution as input, ordinary GE can be composed with back-substitution to form one MMG that solves systems of linear equations.

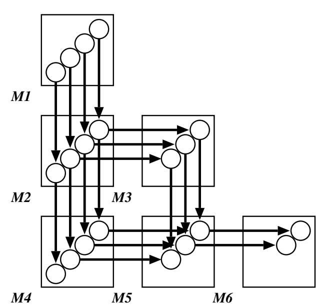
The 3D MMG can then be transformed into one of many possible systolic computational graphs by grouping the primitive nodes along one of the three axes by rectangular prisms of base size $p \times q$, where p is a factor of the projected width size and q is a factor of the projected length. Currently, MAMACG considers only a 1×1 prism.

The set of primitive nodes and links within one group is called a systolic computational cell and the set of systolic computational cells is called a systolic computational graph. Three possible resulting systolic computational graphs of rank-3 ordinary GE are shown in Figure 9, 10, and 11. The choice of the grouping axis and the dimension of the



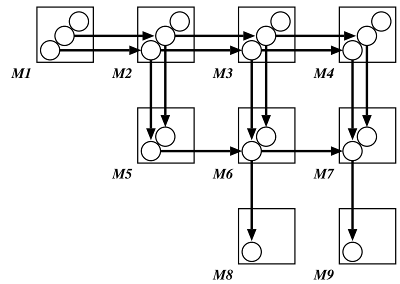
Twelve systolic computational cells were formed from the 3D MMG for GE after the projection of a 1×1 prism along the Z axis. Cell $M12$ contains computational nodes located at $\langle 3,4,1 \rangle$, $\langle 3,4,2 \rangle$ and $\langle 3,4,3 \rangle$; cell $M11$ contains computational nodes located at $\langle 2,4,1 \rangle$ and $\langle 2,4,2 \rangle$; and cell $M10$ contains a node located at $\langle 1,4,1 \rangle$.

Figure 9: Systolic computational graph generated by grouping along the Z-axis



Six systolic computational cells, labeled $M1, \ldots, M6$, were formed after the projection of a 1×1 prism along the Y axis. Cell $M1$ contains computational nodes located at $\langle 1,1,1 \rangle$, $\langle 1,2,1 \rangle$, $\langle 1,3,1 \rangle$, and $\langle 1,4,1 \rangle$; cell $M2$ contains computational nodes located at $\langle 2,1,1 \rangle$, $\langle 2,2,1 \rangle$, $\langle 2,3,1 \rangle$, and $\langle 2,4,1 \rangle$; and cell $M3$ contains nodes located at $\langle 2,2,2 \rangle$, $\langle 2,3,2 \rangle$, and $\langle 2,4,2 \rangle$.

Figure 10: Systolic computational graph generated by grouping along the Y-axis



Nine cells, labeled $M1, \ldots, M9$, were formed after the projection of a 1×1 prism along the X axis. Cell $M1$ contains computational nodes located at $\langle 1,1,1\rangle$, $\langle 2,1,1\rangle$ and $\langle 3,1,1\rangle$; cell $M2$ contains nodes located at $\langle 1,2,1\rangle$, $\langle 2,2,1\rangle$, and $\langle 3,2,1\rangle$; and cell $M3$ contains nodes located at $\langle 1,3,1\rangle$, $\langle 2,3,1\rangle$ and $\langle 3,3,1\rangle$.

Figure 11: Systolic computational graph generated by grouping along the X-axis

rectangular prism affect the performance and cost of the derived systolic computational graph, as discussed in [7].

5 Summary

It is known that systolic systems for solving Gaussian Elimination (GE) that include pivoting have much higher timing complexity than their nonpivoting counterparts. It is also known that among alternative techniques, Givens rotations and Gram-Schmidt require costly and complex computations. Unfortunately, GE without pivoting and matrix triangularization with pairwise pivoting fail to work for a common kind of degeneracy, where good (nonsingular) input matrices have bad (singular) leading principal submatrices.

This paper has shown how an input randomization technique can be used to avoid the problems of degeneracy, and implement GE without pivoting as a systolic system that solves systems $A\mathbf{x} = \mathbf{b}$ of linear equations. The systolic randomization unit, basically the DFT, is efficient and fits nicely with the systolic GE and back substitution units. Experiments have indicated that, although the deterioration in numerical stability can be significant, our approach gives acceptable errors for well-conditioned input matrices. It is also possible that pairwise pivoting, or some other local strategy, will prove ameliorative in practice.

Generally, we have tried to show that input randomization can be a useful method of regularizing complex computations. The resulting algorithms are more amenable to parallel architectures like systolic arrays, and more amenable to systematic derivation using current-generation tools like Mamacg. Thus the combination of tools and techniques surveyed in this paper can be adapted for automatic derivation of large-scale systolic systems.

References

[1] W.M. Gentleman and H.T. Kung. Matrix triangularization by systolic arrays. *Proceedings of the SPIE - The International Society for Optical Engineering*, 298:19–

26, August 1981.

[2] H.T. Kung. Why systolic architectures? *Computer*, 15(1):37–46, January 1982.

[3] D. Lê, M. Ercegovac, T. Lang, and J. Moreno. MAMACG: a tool for automatic mapping of matrix algorithms onto mesh array computational graphs. In J. Fortes, E. Lee, and T. Meng, editors, *Proceedings of the International Conference on Application Specific Array Processors*, pages 511–525. IEEE Computer Society Press, August 1992.

[4] G.M. Megson. A systolic helix for matrix triangularisation with partial pivoting. *Parallel Computing*, 14(2):199–206, June 1990.

[5] G.M. Megson. A fast Faddeev array. *IEEE Transactions on Computers*, 41(12):1594–1600, December 1992.

[6] G.M. Megson. *An introduction to systolic algorithm design*. Oxford science publications. Oxford University Press, New York, 1992.

[7] J.H. Moreno and T. Lang. Matrix computations on systolic-type meshes: an introduction to the multimesh graph method. *Computer*, 23(4):32–51, April 1990.

[8] D.S. Parker. Random butterfly transformations with applications in computational linear algebra. Technical Report CSD-950023, UCLA Computer Science Department, 1995.

[9] D.S. Parker and B. Pierce. The randomizing fft: an alternative to pivoting in Gaussian elimination. Technical Report CSD-950037, UCLA Computer Science Department, 1995.

[10] Y. Robert. Solving dense linear systems on systolic architectures. In E.F. Deprettere and A.-J. Van Der Veen, editors, *Algorithms and parallel VLSI architectures: lectures and tutorials presented at the International Workshop 1990*, volume A, pages 363–401, Amsterdam, 1991. Elsevier.

[11] L.N. Trefethen and R.S. Schreiber. Average-case stability of Gaussian elimination. *SIAM Journal on Scientific and Statistical Computing*, 11(3):335–360, 1990.