# Matrix Inversion in $O(\log n)$ on a Scan-Enhanced Reconfigurable Mesh Computer

Alberto Moreira Bryant W. York *

College of Computer Science
Northeastern University
Boston, Massachusetts 02115

## 1 Abstract

With the arrival of the current generation of fast processor chips and improved interconnect technology, low-cost 3-dimensional reconfigurable mesh computers have become more feasible. They could present an attractive price/performance option to large supercomputers and small clusters of workstations. In 1976 Csanky introduced a parallel algorithm for matrix inversion which executed in $O(\log^2 n)$ steps on $n^4$ processors. Csanky's algorithm was designed for a CREW PRAM. More recently, Leighton produced an implementation of Csanky's algorithm for $n$ meshes of trees which achieves $O(\log^2 n)$ steps on $4n^4 - 3n^3$ processors.

In this work we show that a 3-dimensional reconfigurable mesh with $n^4$ processors can perform Csanky's matrix inversion algorithm in $O(\log^2 n)$ steps. With hardware assist capable of executing Blelloch's +-scans along one privileged dimension in $O(1)$ time, Csanky's algorithm for matrix inversion can be performed in $O(\log n)$ time on a 3-dimensional reconfigurable mesh with $n^4$ processors.

*This author's research was partially supported by ARPA grant MDA-972-93-1-0023

## 2 Introduction

### 2.1 Historical Perspective

It has been known for some time that it is possible to compute the inverse of a non-singular square matrix using the Hamilton-Cayley theorem. The theorem dates from 1853 (W.R.Hamilton) and 1858 (A.Cayley) and is mentioned in many books on Linear Algebra, see for example [7]. The method requires that the coefficients of the matrix's characteristic polynomial be computed prior to the application of the theorem. One popular method for computing such coefficients is attributed to an 1840 paper by Leverrier [15]: build up a lower triangular matrix from the traces of increasing powers of the original matrix and multiply its inverse by a column vector of the traces. In 1949, Frame [8] presented a recursive algorithm to invert a matrix based on successively computing the coefficients of the characteristic polynomial. A good description of both Leverrier's work and the Hamilton-Cayley theorem can be found in section 47 [7].

### 2.2 Related Work

In 1976 Csanky [6] developed a parallel version of the Leverrier-Hamilton-Cayley matrix inversion algorithm on a computing model similar to the modern CREW PRAM. His algorithm runs in $O(\log^2 n)$ time and requires $O(n^4)$ processors. More recently Leighton [14] showed an implementation of Csanky's algorithm where an $n \times n$ matrix can be inverted using $n$ three-dimensional meshes of trees, each with $4n^3 - 3n^2$ processors. The time complexity of this implementation is also $O(\log^2 n)$. Leighton points out that Csanky's algorithm always produces an exact so-

lution but is not numerically stable. There are methods based upon Newton iteration which are more stable and use only $O(n^3)$ processors but do not always find an exact solution.

In recent years several authors have proposed reconfigurable architectures for parallel computers. Current reconfigurable parallel architectures generally follow the SIMD hardware model, where a host computer broadcasts instructions to a (back end) collection of processors connected by an interconnection network. What is unique to reconfigurable architectures is that a portion of the instruction is used to "reconfigure" the network (processor port connections) possibly on every instruction cycle. Examples are CHiP [19], meshes with broadcast buses [11] [20], Polymorphic Torus [16] and PARBS [21]. More recently, Miller, Kumar, Reisis and Stout [17] proposed a mesh with a reconfigurable bus which captures features of many other reconfigurable architectures under a unified model.

Several algorithms for three or higher dimensional reconfigurable meshes have been published. Jang and Prasanna [10], Nigam and Sahni [18], and Wang, Chen and Lin [22] proposed algorithms to sort $n$ values in $O(1)$ time on an $O(n^3)$ 3D reconfigurable mesh. Wang and Chen [21] showed a number of basic graph algorithms for a 3-dimensional reconfigurable mesh, including connected cycles, transitive closure, and multiplication of boolean matrices, all in $O(1)$ time. Champion and Rothstein [3] designed an algorithm for solving the longest common subsequence problem in $O(1)$ time on a 3-dimensional PARBS. Chung [5] extended 2-dimensional parallel prefix computations into higher dimensional reconfigurable meshes. Kao, Horng, Wang and Tsai [13] proposed several $O(1)$ basic algorithms for k-dimensional reconfigurable meshes, such as prefix sums and matrix transpose, sorting n elements in $O(1)$ time using $O(n^{5/3})$ processors and finding the kth smallest element of n m-bit integers in $O(m)$ time.

Low-cost three-dimensional mesh computers are now becoming feasible with the introduction of the SHARC single-chip computer [2]. This chip includes 512 kilobytes of on-chip static memory, a peak performance of 120 MFLOPS and six ports that allow it to be connected in a 3-dimensional mesh configuration. Boards with 32 or 64 SHARC processors that plug into a personal computer bus will soon be available for sale [9].

## 2.3 Background on Scans

The scan operator was first introduced in the late 1950s by Iverson in his APL programming language [12]. Given a vector $V$ the result of its *sum-scan*, denoted in APL by $+\backslash V$, is another vector containing the partial sums of the elements taken from left to right. The "backslash" is treated as an operator which applies the "+" function to pairs of elements. The pairs of elements are selected in the following way: The first pair of elements consists of the "identity" for the chosen binary function (zero in the case of +) and the first element of the vector. The result of that binary operation becomes the first component of the result vector. It is also used as one of the operands along with the next vector element in the next application of the "+" function. This process is continued until the elements of the vector are exhausted. The example below shows a simple *sum-scan* of the 8-vector consisting of the first 8 positive integers. The scan operator is conceptually well-defined for any binary associative function with identity.

$$V = 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8$$
$$+\backslash V = 1\ 3\ 6\ 10\ 15\ 21\ 28\ 36$$

Blelloch [1] developed the concept of scans as primitive parallel operations, and stated a number of basic parallel algorithms in terms of scans. He presented a hardware design that implements scans along a row of processors in $O(1)$ time and showed that when this hardware assist is present, many scan-based algorithms experience a reduction of $\log N$ in their asymptotic bounds when compared to non-scan algorithms. It is well known that, without special hardware assist, a scan operation on an $N$-element vector can be done in $O(\log N)$ parallel steps on an $N$ processor PRAM.

Blelloch also utilized the notion of "segmented scans" where a vector of boolean values is used to specify segments within the original vector and the scan is applied to each segment independently. With these and other tools, he presented several basic algorithms in terms of scans, showing that asymptotic speed bounds can be improved if scan operations can be implemented in constant time. He showed a hardware-level state machine design for scans and estimated that on a 64x64 mesh with a 100ns clock time, a scan on a 32-bit field would require 5 microseconds.

This time is in the order of magnitude of a complex machine instruction such as multiply or divide.

### 2.4 Summary of this Paper

In this paper we show that Csanky's algorithm can be implemented on a 3-dimensional reconfigurable mesh in $O(\log^2 n)$ steps on $n^4$ processors. If the reconfigurable mesh has hardware assist for scans along one of its dimensions, then the algorithm runs in $O(\log n)$ time.

We state some basic 3-dimensional data movement algorithms for a reconfigurable mesh, and use them to show that without scan enhancement on an $n \times n \times n$ reconfigurable mesh (1) two $n \times n$ matrices can be multiplied in $O(\log n)$ time, and (2) an $n \times n$ triangular matrix can be inverted in $O(\log^2 n)$ time, while in an $n^2 \times n \times n$ mesh (3) it is possible to find the first $n$ powers of an $n \times n$ matrix in $O(\log^2 n)$ time. As a consequence, a 3-dimensional reconfigurable mesh without scan-assisting hardware can perform Csanky's algorithm within his original time and processor bounds.

We then consider a 3-dimensional scan-enhanced reconfigurable mesh and show that multiplication of two $n \times n$ matrices can be done in $O(1)$ time. Hence inverting an $n \times n$ triangular matrix and computing $n$ increasing powers of an $n \times n$ matrix can be done in $O(\log n)$ time. This lowers the bounds for a full matrix inversion to $O(\log n)$ time on $n^4$ processors.

## 3 Multidimensional Reconfigurable Meshes

An n-dimensional Reconfigurable Mesh is an n-dimensional array of processors arranged as a hyperrectangle of dimensions $d_1 \times\ d_2 \ldots \times d_n$ where each processor is connected to its $2n$ neighbors. Two processors $P_{d1,\ldots,dn}$ and $P_{e1,\ldots,en}$ are neighbors if $d_i = e_i$ for all $1 \leq i \leq n$ but one, say, $k$, and either $|d_k - e_k| = 1$ or $|d_k - e_k| = n - 1$. In the first case the connection is called a direct connection, while in the second case it is a wraparound connection.

We assume a coordinate space where the mesh is "stenciled" along each dimension. Considering an $n \times \ldots \times n$ mesh and $1 \leq i \leq n$, then $P_{d_1,\ldots,d_i,\ldots,d_n}$ and $P_{e_1,\ldots,e_i,\ldots,e_n}$ are the same processor if whenever $d_i \neq e_i$ there exists a finite integer $k_i$ such that $e_i = k_i n + d_i$. This stresses the fact that wraparound connections are also neighbor-to-neighbor connections when viewed in a more general way and allows us to see the mesh as a much larger virtual mesh.

Maresca and Li's polymorphic torus [16] is used as a reference architecture, and is extended into multiple dimensions. Each processor has $n$ internal buses, $n$ input ports and $n$ output ports. Connections to neighbors are assumed to be bi-directional and half-duplex. Each dimension has one internal bus and two ports, and neighboring ports of neighboring processors are connected.

Reconfiguration is attained by allowing each port to optionally connect to each of the $n$ internal buses, establishing data paths along a dimension or across into another dimension. Each internal bus has a control register with $2n$ bits, one per existing port, allowing each port to connect to any of the internal buses. If a port connects to two or more buses at the same time, those buses become interconnected. Figure 1 shows a 2-dimensional reconfigurable mesh processor, while figure 2 shows a 3-dimensional processor. Figure 3 shows an $8 \times 8$ 2-dimensional reconfigurable mesh.

For example, a 2-dimensional torus would have two buses, let's call them Longitudinal and Transversal. Each bus has a control register with four bits, corresponding to 4 gates that allow North, South, East and West ports to optionally connect to the bus. Establishing row and column buses requires every processor to connect its North and South ports to the Transversal bus, and its East and West ports to the Longitudinal bus. Establishing NE and SW L-shaped connections is done by having a processor connect its North and East port to the processor's Transversal bus and its South and West ports to the processor's Longitudinal bus.

## 4 Csanky's Algorithm

In his 1976 paper, Csanky [6] described a parallel implementation of the well known mathematical algorithm to compute the inverse of a matrix using the Cayley-Hamilton theorem and Leverrier's lemma. Csanky used a computing model which is essentially equivalent to a CREW PRAM. We summarize Csanky's algorithm for the benefit of the reader:

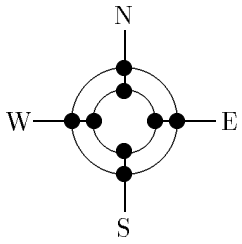


Figure 1: A 2-dimensional reconfigurable processor. The straight lines stand for the four ports. The circles represent the two buses, while the solid dots show the reconfiguration switches that can be opened or closed.

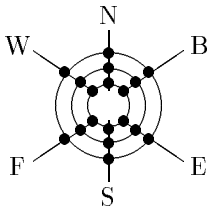


Figure 2: A 3-dimensional reconfigurable processor. The ports are North, South, East, West, Front and Back.

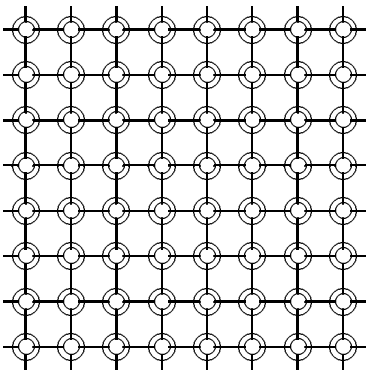

Figure 3: A 2-dimensional reconfigurable mesh. The wraparound connections were omitted for the sake of clarity.

1. For an $n \times n$ matrix $A$, compute all powers $A^k$, $1 \leq k \leq n$.

2. Compute the traces $t_k$ of each $A^k$ by summing its diagonal elements.

3. Leverrier's lemma states that

$$\begin{bmatrix} 1 & & & & \\ t_1 & 2 & & & \\ t_2 & t_1 & & & \\ \vdots & \vdots & & & \\ t_{n-1} & t_{n-2} & \ldots & t_1 & n \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{bmatrix} = - \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ \vdots \\ t_n \end{bmatrix}$$

   From which the coefficients $c_k$ of $A$'s characteristic equation are computed as $c = T^{-1}t$, where T is the above matrix of traces.

4. Applying the Hamilton-Cayley theorem the inverse $A^{-1}$ is:

$$A^{-1} = -(A^{n-1} + c_1 A^{n-2} + \ldots + c_{n-1} I)/c_n$$

Step 1 takes $O(\log^2 n)$ steps and requires $O(n^4)$ processors. Step 2 is done in $O(\log n)$ time on $O(n^2)$ processors. For step 3, inversion of the triangular matrix $T$ takes $O(\log^2 n)$ steps on $O(n^3)$ processors. Step 4 can be done in $O(\log n)$ time on $O(n^3)$ processors. This gives an $O(\log^2 n)$ time complexity on $O(n^4)$ processors.

Leighton [14] proposed an implementation of Csanky's algorithm for $n$ 3-dimensional meshes of trees that runs in $O(\log^2 n)$ time. Each mesh of trees requires $4n^3 - 3n^2$ processors. Leighton's implementation computes $A^k$ in $O(\log^2 n)$ time, computes the traces $t_k$ of each of these matrices, and uses a 3-dimensional mesh of trees to invert the $T$ matrix in $O(\log^2 n)$ time by partitioning the matrix as follows:

$$T = \begin{bmatrix} T_1 & 0 \\ T_3 & T_2 \end{bmatrix}$$

Therefore

$$T^{-1} = \begin{bmatrix} T_1^{-1} & 0 \\ X & T_2^{-1} \end{bmatrix}$$

where $X = -T_2^{-1} T_3 T_1^{-1}$. This method is amenable to a recursive implementation where $T$ is inverted in $\log n$ steps, each requiring two matrix multiplications.

# 5 3-Dimensional Data Movement on the Reconfigurable Mesh

We will now define some basic 3-dimensional matrix movement operations, which will be used as building blocks of our matrix algorithms. These operations are all executed in a constant number of steps because their implementations take advantage of the reconfigurable buses and circuit switching technology. Let $f$ be a data movement function such that $f : A \rightarrow A$ maps all processors in the set A in a one-to-one fashion, then $f$ will be executed in $O(1)$ time if two conditions hold: (1) for every processor in A we can set up the reconfigurable buses to define a path to its image under $f$, and (2) all paths from processors to their images under $f$ are mutually disjoint.

Assume we have an $n \times n \times n$ mesh with a superimposed 3-dimensional coordinate system. This coordinate system will have three axes: $r$ (for row), $c$ (for column) and $p$ (for plane). Assume we have an $n \times n$ matrix $M$ with one element per processor in a plane $P$. If $m_{11}$ is stored in processor $P_{RCP}$ then $m_{rc}$ is contained in processor $P_{(R+r-1)(C+c-1)P}$. The same element can be expressed in "local" coordinates, that is, relative to the top left corner of the matrix, as $P_{rc0}$.

**Definition 1**: A forward rotation around the row axis $r$ moves every matrix element from processor $P_{rc0}$ to processor $P_{r0c}$ (counterclockwise rotation) or from $P_{rc0}$ to $P_{rnc}$ (clockwise rotation). If before the rotation all elements were in plane $rc$, after the rotation all elements are in plane $rp$. The matrix can be similarly rotated around the column axis or plane axis. See Figure 4 for an example.

**Definition 2**: A reverse rotation is one where the rows along the rotation axes are transposed. In other words, a reverse rotation is a rotation where the pivot axis is translated while the rotation is being performed. Reverse rotating around the row axis moves elements from $P_{rc0}$ to $P_{r0(n-c)}$ (clockwise) or to $P_{rn(n-c)}$ (counterclockwise). The matrix can also be reverse rotated around the column axis or plane axis. A reverse rotation is equivalent to a rotation followed by a translation, see Figure 5 for an example.

**Definition 3**: A translation copies the entire matrix along a given axis. Translating along the row axis copies elements from $P_{rcp}$ into $P_{(r+t)cp}$, where $t$ is the translation distance. Translating the matrix along the column axis copies $P_{rcp}$ into $P_{r(c+t)p}$. A broadcast occurs when a matrix is simultaneously translated into all planes perpendicular to a given axis. A restricted broadcast causes the matrix to be translated only within the bounds established by the reconfiguration hardware. See Figure 6 for an example.

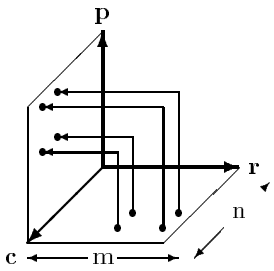


Figure 4: Rotation of an $m \times n$ matrix around the c-axis.

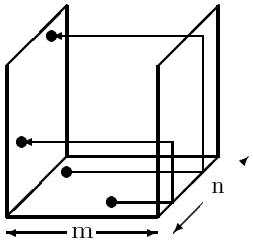


Figure 5: Reverse Rotation of an $m \times n$ matrix around the c-axis.

**Definition 4**: A matrix roll consists of a rotation followed by a broadcast. A reverse roll consists of a reverse rotation followed by a broadcast. For example, a matrix occupying plane $rc$ would be rotated into plane $cp$ and then broadcast along all planes perpendicular to dimension $r$. See figure 7 for an example.

**Definition 5**: A matrix shift by $(t, u, v)$ consists in moving each element from $P_{rc0}$ to $P_{(r+t)(c+u)v}$.

**Lemma 1**: In an $n \times n \times n$ 3-dimensional reconfigurable mesh, translation and broadcast of an $n \times n$ matrix can be done in O(1) time.

Proof: Assume a matrix on plane $rc$ to be translated along the $p$-axis. We can establish buses along the $p$ dimension, so that each $P_{rcp}$ can be broadcast along

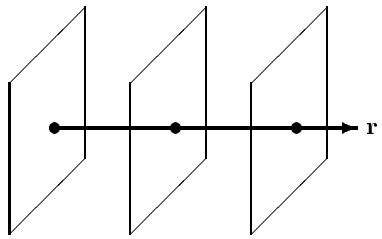


Figure 6: Translation of a matrix along an axis.

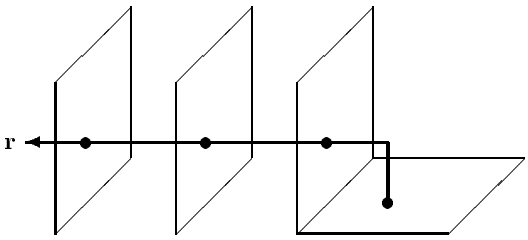


Figure 7: Rolling a matrix.

its respective $p$ bus and be picked up as $P_{rc(p+t)}$ by every $p+t$ that is a receiver of the broadcast.

**Lemma 2**: In an $n \times n \times n$ 3-dimensional reconfigurable mesh, forward or reverse rotation of an $n \times n$ matrix can be done in O(1) time.

Proof: We must forward rotate the matrix around its row axis. Using local indexes, a rotation around the row axis copies each matrix element $P_{rc0}$ into its rotated image $P_{r0c}$. For each $P_{rc0}$ we can establish a 2-step path: a bus along the plane axis from $P_{rc0}$ to $P_{rcc}$, followed by another bus along the column axis $P_{rcc}$ to $P_{r0c}$. For each original matrix element $P_{rc0}$ these buses are unique; all elements $P_{rc0}$ can be moved in parallel, either in one or in two steps depending on whether or not the reconfiguration hardware allows 90-degree turns.

Reverse rotation is similar. We now have to move elements $P_{rc0}$ to $P_{r0(n-c)}$. This requires one p-axis bus from $P_{rc0}$ to $P_{rc(n-c)}$, and a c-axis bus from $P_{rc(n-c)}$ to $P_{r0(n-c)}$. These lines are also unique for each element, and all elements can be moved in parallel.

**Lemma 3**: In an $n \times n \times n$ 3-dimensional reconfigurable mesh an $n \times n$ matrix can be rolled in $O(1)$ time.

Proof: Forward rotation, reverse rotation and broadcast are all $O(1)$ operations.

**Lemma 4**: In an $n \times n \times n$ 3-dimensional reconfigurable mesh an $n \times n$ matrix can be shifted in $O(1)$ time.

Proof: To shift a matrix originally in the $rc$ plane, perform the following steps: Rotate the matrix around the row axis, translate it to the desired row, rotate the matrix around the plane axis, translate it to the desired column, rotate it around the plane axis and roll it into the desired plane. Each of these operations is done in $O(1)$ time.

The lemmas above show that it is possible to move a matrix around a reconfigurable mesh in a wide variety of ways in $O(1)$ asymptotic time. This flexibility allows us to design fast algorithms.

# 6 Matrix Multiplication

**Theorem 1**: In an $n \times n \times n$ reconfigurable mesh an $R \times P$ and a $P \times C$ matrix can be multiplied in $O(logP)$ time, for all $1 \leq R \leq n$, $1 \leq C \leq n$ and $1 \leq P \leq n$.

Proof: The algorithm is similar to the ones used for hypercubes or meshes of trees, see for example Leighton [14], section 2.4.2.

Assume an $R \times P$ matrix $A$ in plane $rp$, and a $P \times C$ matrix $B$ in plane $pc$. Assume also that elements $A_{00}$ and $B_{00}$ are located at the same processor; if they are not, the matrices can be shifted and rotated into position in $O(1)$ time according to lemmas 1-4. We have now matrices A and B making up two orthogonal planes of a coordinate system $rcp$. Broadcast matrix $A$ along the c-axis and broadcast matrix $B$ along the plane r-axis; this is done in $O(1)$ time according to lemma 1.

At this point, every processor $P_{rcp}$ has both $A_{rp}$ and $B_{pc}$; every row parallel to the $p$ axis contains all the pairs required to compute $C_{rc} = \sum_{p=1}^{n} A_{rp} B_{pc}$. All products are computed in parallel, and +-scan along the p-axis will compute all $C_{rc}$ in parallel in $O(\log P)$ time. The product matrix sits now in plane $rc$, from where it can be moved by shifts and/or rotations back to its required destination in $O(1)$ time.

**Theorem 2**: In a $n \times n \times n$ reconfigurable mesh assisted by scan-optimizing hardware along all buses in one dimension, an $R \times P$ and a $P \times C$ matrix can

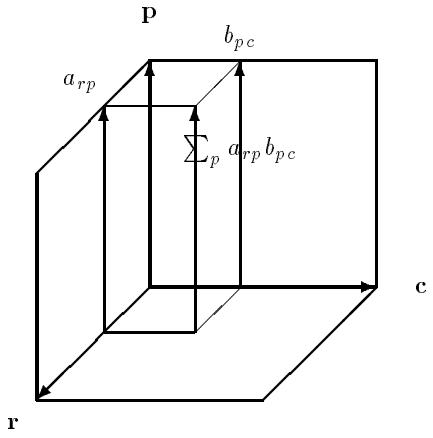


Figure 8: Matrix Multiplication on the 3-dimensional Reconfigurable Mesh. Matrix A lies on the rp plane while matrix B is in the pc plane. The products are added along the p axis and the result appears in the rc plane.

be multiplied in $O(1)$ time.

Proof: Assume that the scan-optimizing hardware is capable of performing +-scans alongside the $p$-axis in $O(1)$ time. The matrices being multiplied can be moved to planes rp and pc in in $O(1)$ steps according to lemmas 1-4. Data is then broadcasted to the desired locations in $O(1)$ steps. The +-scan along the p axis is now done in $O(1)$ time, leaving the result matrix on the rc plane.

The scan-assisting hardware is independent of the reconfiguration hardware for the reconfigurable buses, and typically acts on the whole bus. To make sure that intermediate "off" processors do not alter the results of a scan, it is possible to set their scanned values to zero. If the scan hardware operates in constant time, the additional processors will not alter the asymptotic running time of the algorithm.

# 7 Inverting a Lower Triangular Matrix

**Theorem 3**: In an $n \times n \times n$ reconfigurable mesh a lower-triangular $n \times n$ matrix can be inverted in $O(\log^2 n)$ steps.

Proof: We partition our matrix $T$ in a similar fashion to Leighton's algorithm ([14], section 2.4.3),

$$T = \begin{bmatrix} A & 0 \\ C & B \end{bmatrix}$$

Where $A$, $B$ and $C$ are $n/2 \times n/2$, and $A$ and $B$ are lower triangular. The inverse is given by

$$T^{-1} = \begin{bmatrix} A^{-1} & 0 \\ X & B^{-1} \end{bmatrix}$$

where $X = -B^{-1}CA^{-1}$. If the matrix is originally on the $rc$ plane, recursively invert $A$ and $B$ in parallel, then do the multiplications: rotate $A^{-1}$ around the $r$-axis, multiply it by $C$, leave the result in the $cp$ plane, multiply this result by $B^{-1}$, leave the result in the rp plane, shift and rotate the resulting matrix into the required plane. By Theorem 1 each multiplication is done in $O(\log n)$time. We have

$$T_n = T_{n/2} + O(\log n)$$

That is, $T_n = O(\log^2 n)$.

**Theorem 4**: In a $n \times n \times n$ reconfigurable mesh assisted by scan-optimizing hardware along all buses in one dimension, a lower triangular $n \times n$ matrix can be inverted in $O(\log n)$ steps.

Proof: Using lemmas 1-4, rotate and translate the matrices in $O(1)$ steps so that the axis common to both matrices is along the optimizing hardware. Each matrix multiplication is then performed in $O(1)$ time, according to theorem 2. Therefore the recurrence becomes

$$T_n = T_{n/2} + O(1)$$

That is, $T_n = O(\log n)$.

# 8 Computing the increasing powers of a matrix

This is the step in the algorithm where a $n^4$ processors are required. Assume that we want to compute increasing powers $A^k$ of a matrix, for $1 \leq k \leq n$. Each $A^k$ is an $n \times n$ matrix, and the $n$ matrix powers require $n$ mesh planes. To perform all the required matrix multiplications in parallel and keep a logarithmic execution time, we need to allow a cube of $n^3$ processors for each matrix plane, so that matrix multiplications can use the algorithm of Theorems 1 and 2. Therefore, we need an $n^2 \times n \times n$ mesh, that is, $n^4$ processors for this sub-algorithm.

**Theorem 5**: In a 3-dimensional $n^2 \times n \times n$ reconfigurable mesh it is possible to compute the first $n$ increasing powers of a matrix $A$ in $O(\log^2 n)$ time.

Proof: Assume that matrix $A$ lies on the $rc$ plane, and that we have $n^2$ planes along the p axis. Assume also that the n powers $A^k$, $1 \leq k \leq n$ will lay on planes parallel to the $rc$ plane, so that $p$ is the scan direction.

The algorithm starts by broadcasting $A$ along the $p$ axis so that every plane has its own copy. Planes $kn$, $0 \leq k \leq n-1$, all have their own copies of matrix A, and each is the boundary of a cube containing $n^3$ processors where matrix multiplications will take place. The reconfigurable buses are used to make the machine look like an $n \times n \times n$ machine where each plane actually contains a subordinate $n \times n \times n$ work structure.

A standard prefix algorithm will generate all powers of $A$ at the correct planes. Each matrix multiplication is performed by rotating the current multiplicand around the r axis and then using the matrix multiplication algorithm described in Theorem 1. The partial products are then rotated back into the $rp$ plane in $O(1)$ steps. The fact that the power matrices are separated from each other by $n$ planes doesn't slow down the rest of the algorithm because the reconfigurable buses can be stretched to bridge over the unused processors.

**Theorem 6**: In a 3-dimensional $n^2 \times n \times n$ reconfigurable mesh endowed with hardware +-scans along one of its axes, it is possible to compute the first $n$ increasing powers of a matrix in $O(\log n)$ time.

Proof: According to Theorem 4, each of the $\log n$ matrix multiplication steps is now done in $O(1)$ steps, including any required rotations and translations.

# 9 Full Matrix Inversion on the Reconfigurable Mesh

Let us now consider the full matrix inversion algorithm. An implementation of Csanky's algorithm for the reconfigurable mesh with +-scans along one dimension executes as follows:

1. Compute all ascending powers $A^k$, $1 \leq k \leq n$. According to Theorems 5 and 6, this is done in $O(\log^2 n)$ in the standard reconfigurable mesh and in $O(\log n)$ if the mesh is scan-assisted.

2. Compute the traces $t^k$ of each $A^k$. This requires each $A^k$ to perform a +-scan along its main diagonal. Because each $A^k$ now lies in a separate plane, all scans are done in parallel in $O(\log n)$ time.

3. Compute the coefficients of $A$'s characteristic polynomial by applying Leverrier's lemma. This requires inverting a lower triangular matrix, which is accomplished via theorems 3 and 4 in $O(\log^2 n)$ steps in the reconfigurable mesh and in $O(\log n)$ time when the mesh is scan-assisted. The matrix-vector multiplication to compute the actual coefficients doesn't affect the asymptotic time of this step.

4. Apply the Hamilton-Cayley theorem to compute the matrix inverse:

$$A^{-1} = -(A^{n-1} + c_1 A^{n-2} + ... + c_{n-1} I)/c_n$$

This requires broadcasting each coefficient $c_i/c_n$ into $A_i$'s plane, multiplying all elements of each $A_i$ by its coefficient in parallel (this is done in $O(1)$ time), and then adding all matrices via a +-scan. The +-scan dominates the asymptotic time, it runs in $O(\log n)$ time in the reconfigurable mesh and in $O(1)$ when the mesh is scan-assisted.

Each step above can be done in $n^3$ processors, except for step 1 which needs $n^4$ processors. This leads to

**Theorem 7**: Inverting an $n \times n$ matrix can be performed in $O(\log^2 n)$ time on a 3-dimensional $n^2 \times n \times n$ reconfigurable mesh, or in $O(\log n)$ time if the mesh includes hardware assists to perform +-scans in $O(1)$ time alongside all buses of one privileged dimension.

# 10 Conclusions

We have shown that a 3-dimensional reconfigurable mesh can perform Csanky's matrix inversion algorithm in $O(\log^2 n)$ time using $n^4$ processors. We have also shown that if scan-assisted hardware is used to allow +-scans to be performed in $O(1)$ time alongside one privileged dimension, the reconfigurable mesh will perform full matrix inversion in $O(\log n)$ time using $n^4$ processors.

Algorithms for 3-dimensional reconfigurable mesh computers are easier to specify when we use a set of primitive data movement operations. Two of the future goals of this work are (1) the definition of a

broader set of basic operations required to efficiently implement matrix algorithms for three- and higher-dimensional reconfigurable machines and (2) the development of an associated programming language.

# References


[1] G. Blelloch, Scans as Primitive Parallel Operations, *IEEE Transactions on Computers*, Vol 38 no 11, 1987.

[2] J. E. Brewer, L. G. Miller, I. H.Gilbert, J. F.Melia, D. Garde and J. E. DeMaris, A Monolithic Processing Subsystem, *IEEE Transactions on Components, Packaging, and Manufacturing Technology - Part B*, Vol. 17, No. 3, August 1994.

[3] G. M. Champion and J. Rothstein, Immediate Parallel Solution of the longest common Subsequence Problem, Proc. International Conference on Parallel Processing, pp. 70-77, 1987.

[4] E. Chu, A. George and D. Quesnel, Parallel Matrix Inversion on a Subcube-grid", *Parallel Computing* 19, pp. 243-256, 1993.

[5] K. L.Chung, Prefix Computations on a Generalized Mesh-Connected Computer with Multiple Buses, *IEEE Transactions on Parallel and Distributed Systems*, Vol 6 No 2, February 1995.

[6] L. Csanky, Fast Parallel Matrix Inversion Algorithms, *SIAM Journal of Computing*, vol 5, pp. 618-623, 1976.

[7] D. K. Faddeev and V. N. Faddeeva, *Computational Methods of Linear Algebra*, W. H. Freeman and Company, San Francisco, 1963.

[8] J. S. Frame, A simple Recursion Formula for inverting a Matrix, *Bulletin of the American Mathematical Society*, **55**, p. 1045, 1949.

[9] Integrated Computing Engines, Inc., Technical Report: The MeshSP Mesh Multiprocessing and the ICE RealTime Engine, 1995.

[10] J. W. Jang and V. K. Prasanna, An Optimal Sorting Algorithm on Reconfigurable Mesh, *Journal of Parallel and Distributed Computing*, 25, pp. 31-41, 1995.

[11] W. M. Gentleman, Some Complexity Results for Matrix Computations on Parallel Processors, *Journal of the ACM*, vol 25, pp. 112-115, 1978.

[12] K. E. Iverson, Notation as a Tool of Thought, 1979 Turing Award Lecture, *ACM Turing Award Lectures: The First Twenty Years*, ACM Press, 1987, pp. 339-389.

[13] T. W. Kao, S. J. Horng, Y. L. Wang and H. R. Tsai, Designing Efficient Parallel Algorithms on CRAP, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, No. 5, pp. 554-560, May 1995.

[14] T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann Publishers, San Mateo, California, 1992.

[15] U. LeVerrier, Sur les variations seculaires des elements des Orbites, *J. Math.*, 1840.

[16] M. Maresca and H. Li, Connection Autonomy in SIMD Computers: A VLSI Implementation, *Journal of Parallel and Distributed Computing*, No 7, 1989, pp. 302-320.

[17] R. Miller, V. K. Prasanna-Kumar, D. I. Reisis and Q. F. Stout, Parallel Computations on Reconfigurable Meshes, *IEEE Transactions on Computers*, Vol 42 No 6, pp. 678-692, June 1993.

[18] M. Nigam and S. Sahni, Sorting $n$ Numbers on $n \times n$ Reconfigurable Meshes with Buses, *Journal of Parallel and Distributed Computing*, 23, pp. 37-48, 1994.

[19] L. Snyder, Introduction to the Highly Parallel Computer, *IEEE Computer*, pp. 47-56, January 1982.

[20] Q. F. Stout, Mesh-Connected Computers with Broadcasting, *IEEE Transactions on Computers*, Vol. C-32 No. 9, September 1983.

[21] B. F. Wang and G. H. Chen, Constant Time Algorithms for the Transitive Closure and Some Related Graph Problems on Processor Arrays with Reconfigurable Bus Systems, *IEEE Transactions on Parallel and Distributed Systems*, Vol 1 No 4, pp. 500-507, October 1990.

[22] B. F. Wang, G. H. Chen and F. C. Lin, Constant Time Sorting on a Processor Array with a Reconfigurable Bus System, *Information Processing Letters*, vol 34 no 4, pp. 187-192, April 1990.