# A Proposed Parallel Architecture for Matrix Triangularization with Diagonal Loading

C. M. Rader
MIT Lincoln Laboratory [*]
Lexington, MA

April 22, 2004

## Abstract

We are given $M$ training samples of $N$-element column vectors in a matrix $X$ and a predefined constant $\lambda$. We want to compute the lower-triangular matrix which is the Cholesky factor of $R = XX^t + \lambda I$ using highly parallel hardware, either using FPGAs or ASICs. Adding $\lambda$ is called diagonal loading. In most adaptive processing applications, diagonal loading is used to reduce the sensitivity of the adaptation to errors due to insufficient sample support and to slight errors in the target model.

Mathematically, we first prefix $\sqrt{\lambda}I$ to $X$ and then we use $N$ size $M+1$ Householder postmultiplication, each carried out in a *virtual* superprocessor. We format the computation so that each Householder operation affects the same number of columns, but with fewer and fewer rows. *Actual* superprocessors each share the work of two virtual superprocessors. This allows each superprocessor to be physically identical with each other, while all are used with 100% efficiency. Data is moved from one superprocessor to another a row at a time.

[*]This work is sponsored by Defense Advanced Research Projects Agency, under Air Force Contract F19628-00-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the author and are not necessarily endorsed by the United States Government.

# Report Documentation Page

*Form Approved*
*OMB No. 0704-0188*

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

| | | |
|---|---|---|
| 1. REPORT DATE<br>**01 FEB 2005** | 2. REPORT TYPE<br>**N/A** | 3. DATES COVERED<br>**-** |
| 4. TITLE AND SUBTITLE<br>**A Proposed Parallel Architecture for Matrix Triangularization with Diagonal Loading** | | 5a. CONTRACT NUMBER |
| | | 5b. GRANT NUMBER |
| | | 5c. PROGRAM ELEMENT NUMBER |
| 6. AUTHOR(S) | | 5d. PROJECT NUMBER |
| | | 5e. TASK NUMBER |
| | | 5f. WORK UNIT NUMBER |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)<br>**MIT Lincoln Laboratory Lexington, MA** | | 8. PERFORMING ORGANIZATION REPORT NUMBER |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | | 10. SPONSOR/MONITOR'S ACRONYM(S) |
| | | 11. SPONSOR/MONITOR'S REPORT NUMBER(S) |

12. DISTRIBUTION/AVAILABILITY STATEMENT
**Approved for public release, distribution unlimited**

13. SUPPLEMENTARY NOTES
**See also ADM001742, HPEC-7 Volume 1, Proceedings of the Eighth Annual High Performance Embedded Computing (HPEC) Workshops, 28-30 September 2004. , The original document contains color images.**

14. ABSTRACT

15. SUBJECT TERMS

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT<br>**unclassified** | b. ABSTRACT<br>**unclassified** | c. THIS PAGE<br>**unclassified** | **UU** | **26** | |

**Standard Form 298 (Rev. 8-98)**
Prescribed by ANSI Std Z39-18

Most of the arithmetic operations in a Householder transformation are simple in two important ways. They can be prescheduled, and they cannot give unbounded results. Hence they are easily parallelized. These simple operations are segregated into two groups. One group is dot-products and the other group uses operations that multiply one row by a scalar and add it to another row. Both these operations allow us to flow data with each column's elements moving only vertically and maintaining their order, a perfect recipe for systolic computation. Each row is used as soon as it is transferred.

The small number of more complicated operations we need, a square root and a few multiplications and divisions, are carried out in a physically separate part of the superprocessor. All the floating point operations are confined to this part of the processor. They don't need to be fast, because we can keep the multipliers and adders 100% occupied by working on several different triangularizations at the same time.

# Proposed Parallel Architecture for Matrix Triangularization with Diagonal Loading

Charles M. Rader

*This work is sponsored by Defense Advanced Research Projects Agency, under Air Force Contract F19628-00-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the author and are not necessarily endorsed by the United States Government.

Sept. 29, 2004

## The Matrix Triangularization Problem

A common task in adaptive signal processing is as follows: We have a set of N training vectors, each with M components. These constitute a matrix X and we need the Cholesky factor, T, of its correlation matrix $R=XX^h + \lambda I$.

Usually $N >> M$.

The cost of the computation is of the order of $M^2N$. If $M^2N$ is large, we will need some parallel computation to keep up with a real time requirement.

## The Matrix Triangularization Problem

A common approach is to premultiply the N by M matrix X by each of a sequence of Householder matrices, one after the other. Most of the operations required are adds and multiplies, and it is straightforward to perform many adds and multiplies in parallel, but the algorithm also requires a few divisions and square roots. These interfere with the efficiency of the use of a parallel array of multipliers and adders.

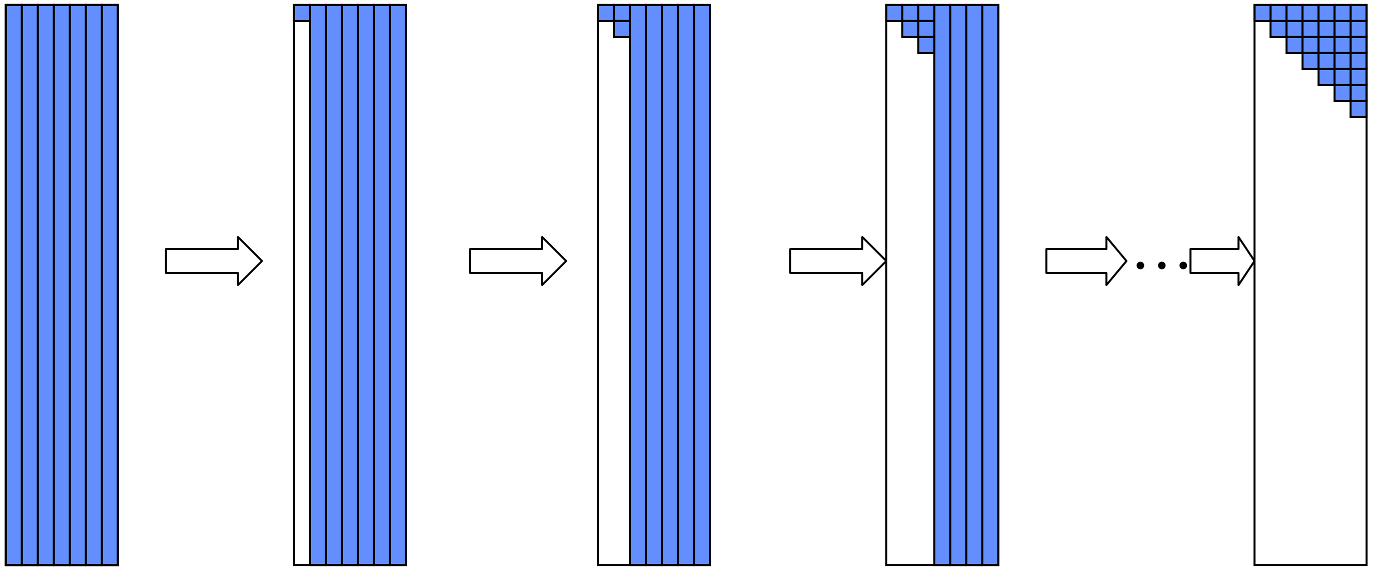
## The Matrix Triangularization Problem

This talk is about an architecture which might be suitable to realize using FPGAs. We have in mind problems with $M \approx 20$ and $N \approx 100$.

FPGAs are now available with approximately 100 built-in multipliers and with the capability to create a similar number of adders. Hence about ten FPGAs should be able to perform about 1000 multiply-adds in parallel.

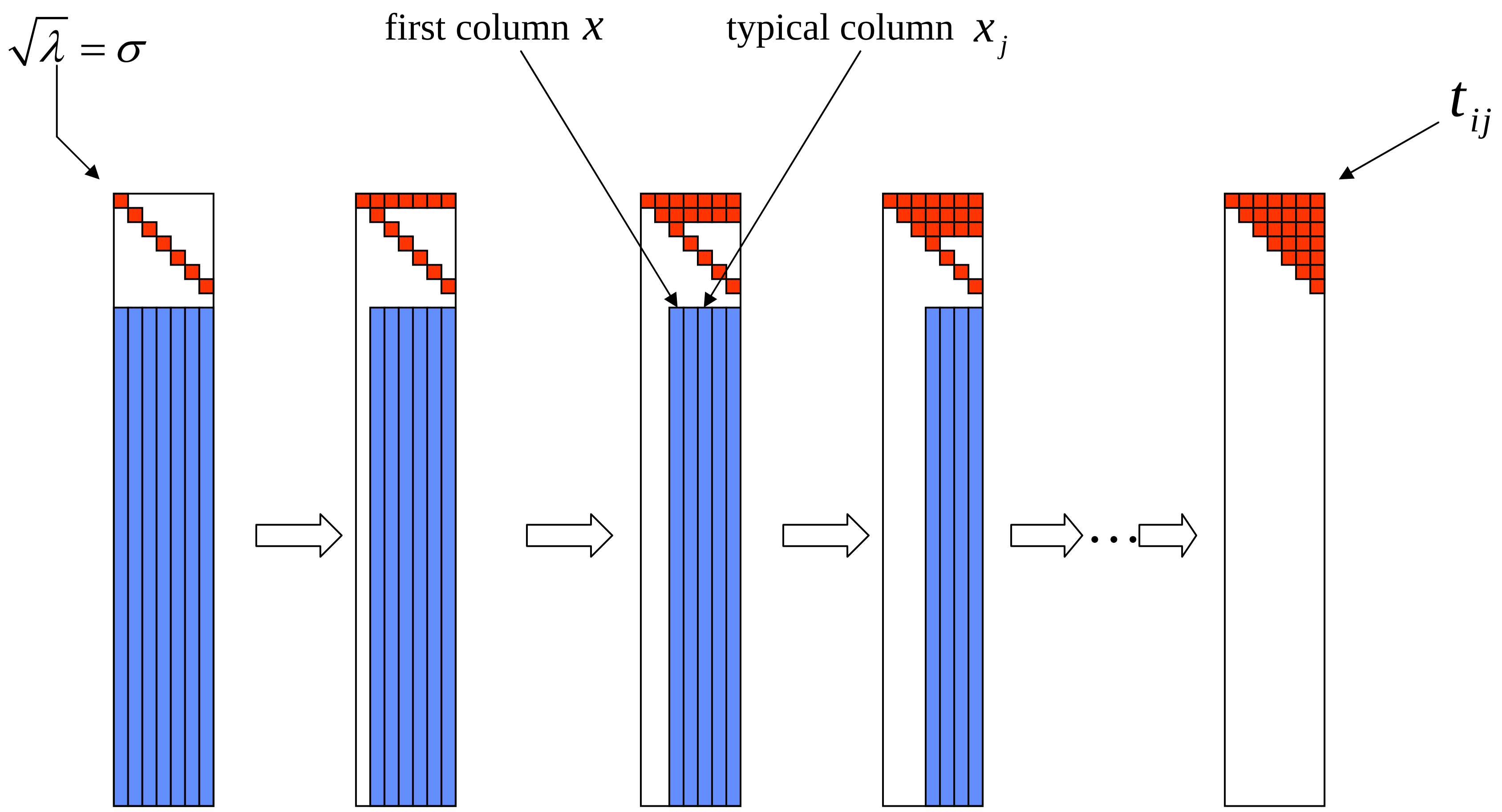
Our architecture should use these 100 multipliers and adders with near 100% efficiency and we desire that all the FPGAs be identical (and, indeed, might later be replaced by custom ASICs.

# Steps to triangularization using unitary matrix premultiplications



MIT Lincoln Laboratory

# Steps to triangularization with diagonal loading



# The Math of Zeroing a Column

$$\phi = x^h x, \phi_j = x^h x_j$$

N operations per column

$$t_{ii} = \xi = \sqrt{\phi + \lambda}, \mu = 1/\xi$$

$$\theta = \frac{1}{\xi(\xi - \sigma)}$$

$$\beta_j = \theta\phi_j$$

1 operation per column

$$x_j' = x_j - \beta_j x$$

N operations per column

$$t_{ij} = \mu\phi_j$$

1 operation per column

## The Math of Zeroing a Column

## Opportunities for parallelism

$$\phi = x^h x, \phi_j = \boxed{x^h x_j}$$

$$t_{ii} = \xi = \sqrt{\phi + \lambda}$$

$$\theta = \frac{1}{\xi(\xi - \sigma)}$$

N multiplications at once
conj($x_i$) • $x_{ij}$ ; i=1,…,N
(times the number of columns)

$$\beta_j = \theta\phi_j$$

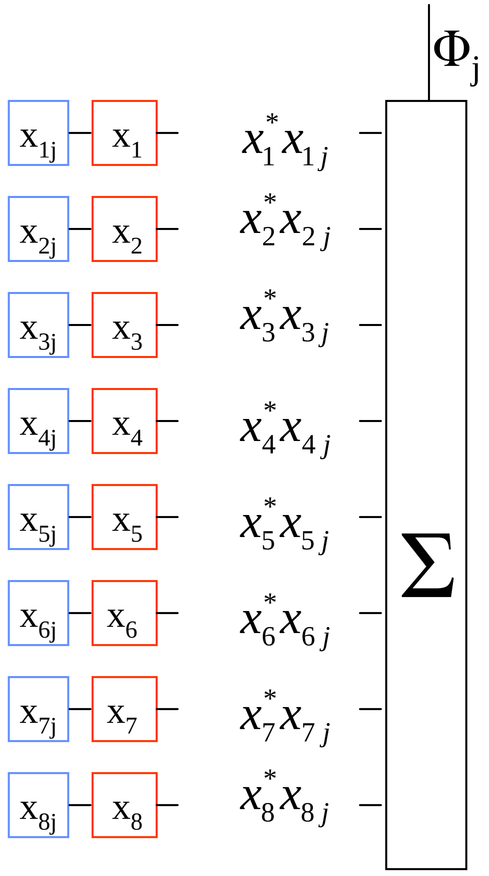
$$x_j' = x_j - \boxed{\beta_j x}$$

$$t_{ij} = \mu\phi_j$$

N multiplications at once
$\beta_j$ • $x_i$ ; i=1,…,N
(times the number of columns minus 1)

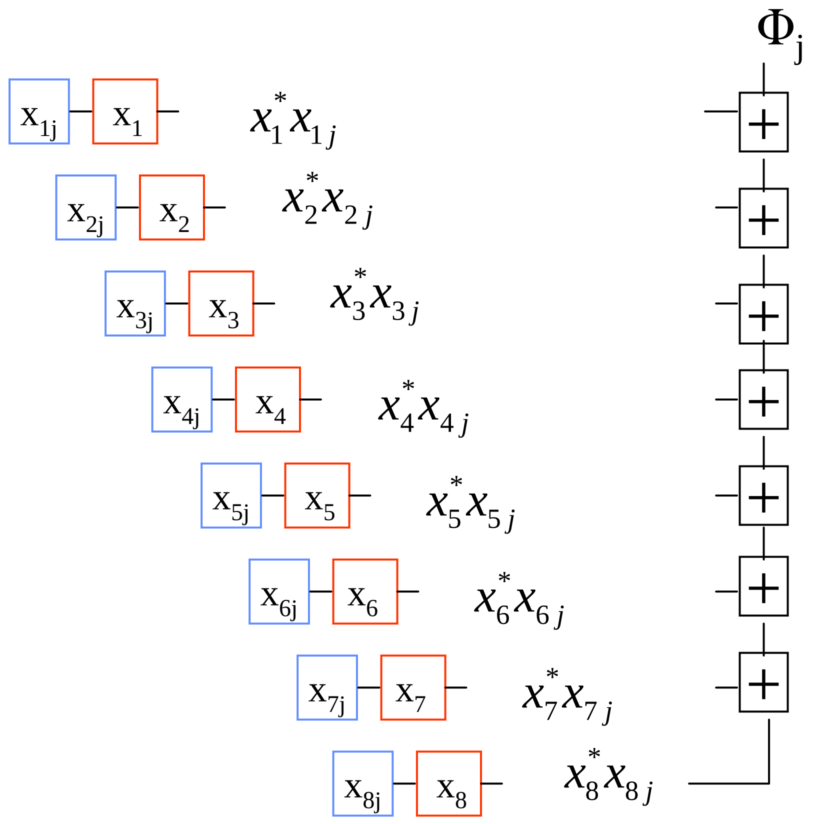
# Front End Processing

## Φ-processor computes $\Phi_j$



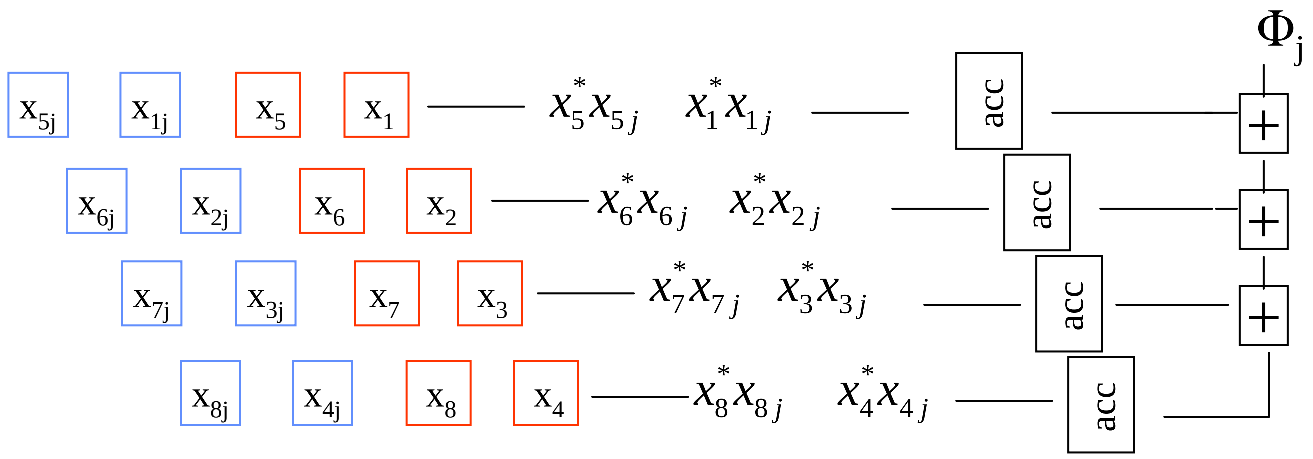
# Skewed Front End Processing

# Computes $\Phi_j$



$x_{1j}$ $x_1$ $x_1^* x_{1j}$
$x_{2j}$ $x_2$ $x_2^* x_{2j}$
$x_{3j}$ $x_3$ $x_3^* x_{3j}$
$x_{4j}$ $x_4$ $x_4^* x_{4j}$
$x_{5j}$ $x_5$ $x_5^* x_{5j}$
$x_{6j}$ $x_6$ $x_6^* x_{6j}$
$x_{7j}$ $x_7$ $x_7^* x_{7j}$
$x_{8j}$ $x_8$ $x_8^* x_{8j}$
$\Phi_j$

# Series-parallel Front End Processing



$x_{5j}$ $x_{1j}$ $x_5$ $x_1$ $x_5^* x_{5j}$ $x_1^* x_{1j}$
$x_{6j}$ $x_{2j}$ $x_6$ $x_2$ $x_6^* x_{6j}$ $x_2^* x_{2j}$
$x_{7j}$ $x_{3j}$ $x_7$ $x_3$ $x_7^* x_{7j}$ $x_3^* x_{3j}$
$x_{8j}$ $x_{4j}$ $x_8$ $x_4$ $x_8^* x_{8j}$ $x_4^* x_{4j}$
acc
acc
acc
acc
+
+
+
$\Phi_j$

## Computing ξ and θ

$$t_{ii} = \xi = \sqrt{\phi + \lambda}$$

Saved as part of the answer

$$\theta = \frac{1}{\xi(\xi - \sigma)}$$

Used in output processor

- **These are needed before output processing can begin.**
- **They are relatively complicated computations and will be computed slowly.**
- **Our aim is to organize the algorithm so that the slow computation of ξ and θ can be buried.**

# Output processor

For each column

$$\beta_j = \theta\phi_j$$

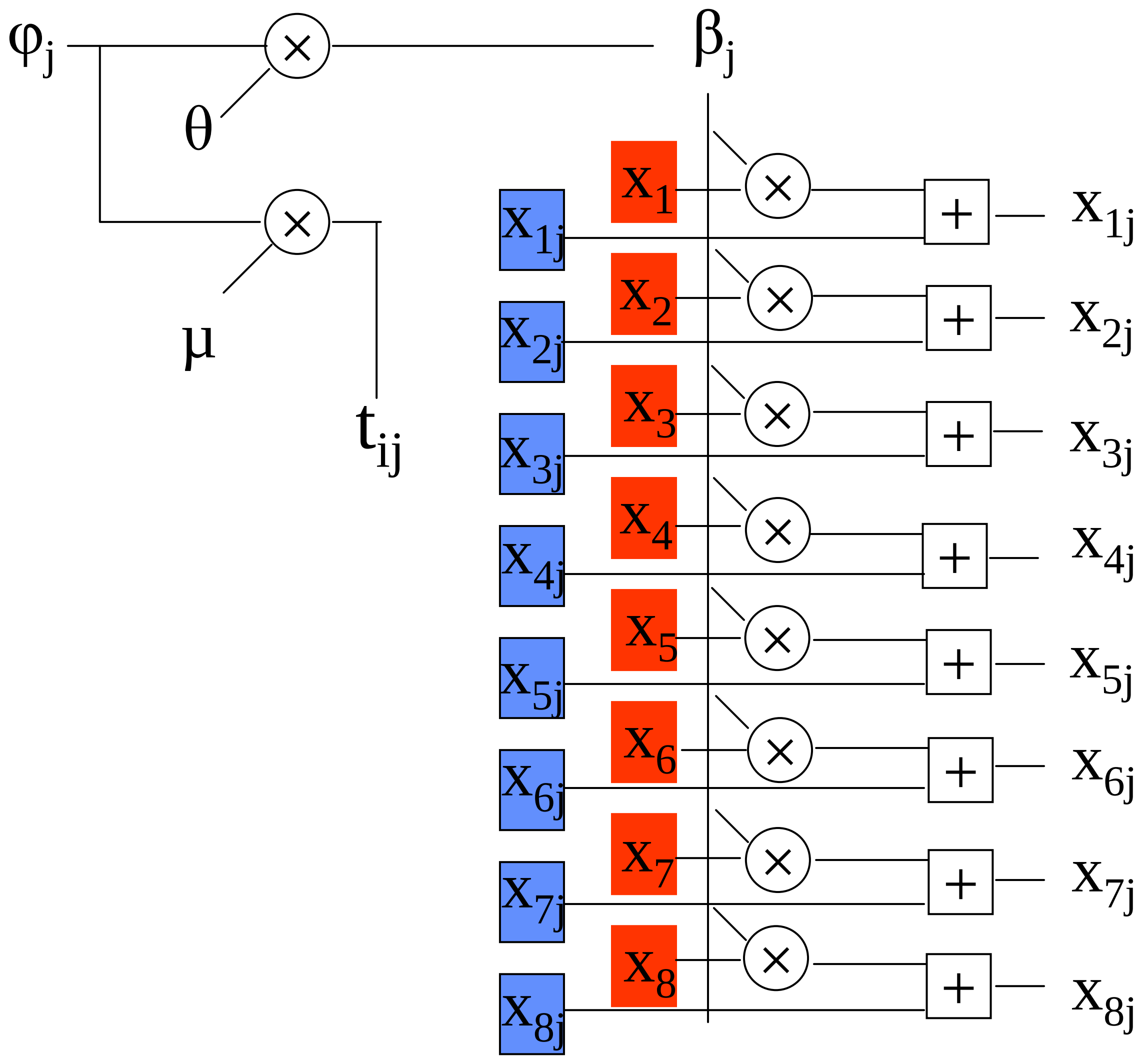
Compute and broadcast to all multipliers

$$x'_j = x_j - \beta_j x$$

One complex multiply per element. The first column was saved and the current column streams by.

$$t_{ij} = \mu\phi_j$$

# Output Processor



# Overview of the column elimination

- There are M columns. The process that eliminates column i accepts M-i+1columns in their normal order and spits out M-i columns.
- Elements move only horizontally and are involved in arithmetic only with other elements on the same horizontal level.
- Sums propagate upward -- $\xi$ , $\theta$, and $\beta_j$ are computed at the upper edge of the processor.
- $\beta_j$ must travel upward from the bottom edge to where it is needed by a multiplier.

## Many Processors

- Let us define a *virtual super-processor.* Its job is to zero out one column.
- Let $\tau$ be the time separation between successive columns presented to the processor input.
- $\tau$ is also the time required to do the multiplies needed for $\Phi_j$ and is the time multiply x by $\beta_j$.
- Then the time that column j spends inside the virtual super-processor is K $\tau$ and most of this is waiting for the computation of $\xi$, $\theta$, and $\beta_j$. (We'll determine K later.)
- We desire that the *virtual super-processor* whose job is to zero out column i+1 be ready for column j as soon as it is computed by the previous virtual super-processor.

## When do columns get where?

| virtual superprocessor | first column enters | column j enters | last column enters |
|---|---|---|---|
| 1 | 0 | (j-1) $\tau$ | (M-1) $\tau$ |
| 2 | (K+1) $\tau$ | (K+j-1) $\tau$ | (K+M-1) $\tau$ |
| 3 | (2K+2) $\tau$ | (2K+j-1) $\tau$ | (2K+M-1) $\tau$ |
| i | ((i-1)K+(i-1)) $\tau$ | ((i-1)K+j-1) $\tau$ | ((i-1)K+M-1) $\tau$ |

## Super-processor sharing

| virtual superprocessor | first column enters | column j enters | last column enters |
|---|---|---|---|
| i | ((i-1)K+(i-1)) τ | ((i-1)K+j-1) τ | ((i-1)K+M-1) τ |
| M+1-i | ((M–i )K+(M-i)) τ | ((M-i)K+j-1) τ | ((M-i)K+M-1) τ |

These two virtual super-processors together process M+1 columns, independent of i, so it is tempting to combine them into one actual super-processor. (M/2) actual super-processors are needed for the whole triangularization.

# Make Super-processors Identical

| virtual superprocessor | first column enters | column j enters | last column enters |
|---|---|---|---|
| i | ((i-1)K+(i-1)) τ | ((i-1)K+j-1) τ | ((i-1)K+M-1) τ |
| M+1-i | ((M–i )K+(M-i)) τ | ((M-i)K+j-1) τ | ((M-i)K+M-1) τ |

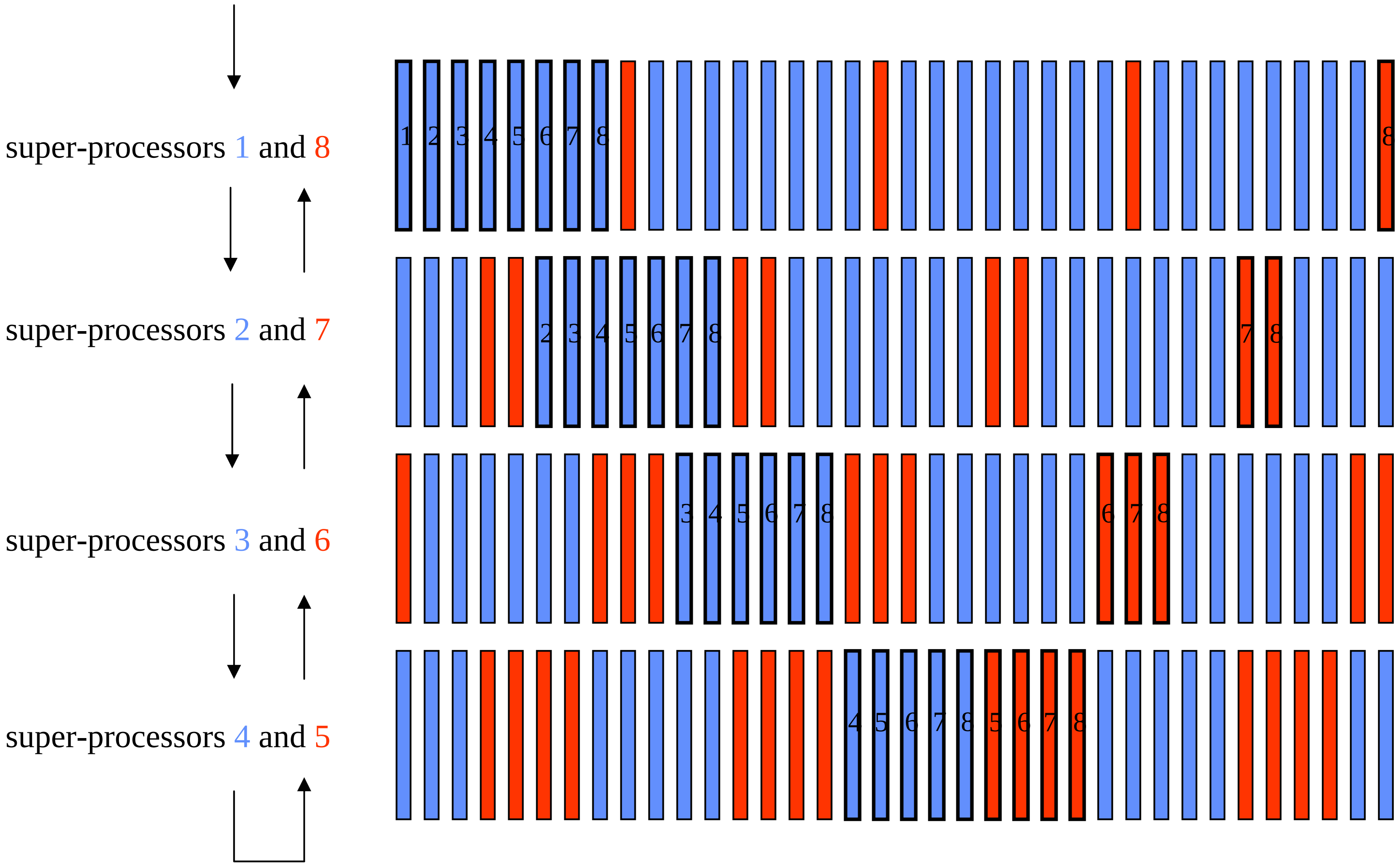
When actual super-processor i accepts its last column from actual super-processor i-1, in the next interval it is ready to accept the first column from actual super-processor i+1, but that must be from an earlier triangularization problem. The M/2 super-processors begin a new triangularization problem every (M+1) τ.

If that column is to be ready, we require $(i-1)K + M \equiv (M-i)(K+1)$ modulo M+1
or

$$(2K+1)i \equiv (M+1)K \equiv 0 \qquad \text{modulo M+1}$$

So we choose K to make 2K+1 = M+1, K=M/2

# Column Timing



## A Dose of Reality

- Problems with word length and scaling
- Problems with input and output

## Problems with word length and scaling

The N elements of input column i have the same total energy as the i elements of the final output for that column, so some element might have dynamic range expansion of up to $\sqrt{N}$.

So we might need floating point. (This is not a result of the architecture – it is intrinsic to the problem.) FPGAs come with efficient built-in multipliers, but not built-in floating point. We don’t know how many floating point multipliers and adders we can get in a single FPGA.

## Problems with input and output

Our architecture has negligible internal control, but requires that data arrive from multiple problems at just the right time, including skewing.

Several problems are active at once and late t-elements from one problem get delivered to the customer after the early t-elements from later problems.

So we will need an interface that transfers data for several "customers" to and from the processing array.

## Summary

We've presented principles for an architecture suitable for realizing matrix triangularization with highly parallel use of multipliers and adders. Identical parts are used and internal control is negligible.

Parallelism comes from working on many independent problems at once. The waiting time for square roots and divisions is buried and does not reduce the efficiency of the use of multipliers.

The architecture will only become practical when FPGAs can realize large numbers of floating point adders and multipliers.