Algorithm Based Fault Tolerance for Numerical Linear Algebra on Multi-core Processors

Soumyendu Raha
(Email: raha@serc.iisc.ernet.in)
SERC, IISc, Bangalore
Significant Probability of Transient Faults

- More components densely packed on to a single chip and in spite of the reliability of the CMOS process the probability of getting transient faults, also called soft errors, on a single chip becomes significant even during the normal life time. (ITRS Roadmap)
  - Probability is about 0.1 at 0.1 micrometer cell pitch for 65 nm CMOS process

- Add to this, more long running applications, such as simulation algorithms in bio molecular dynamics, and design optimization algorithms for integrated nano-engineering systems.
  - Compared to mean time to failure for a single component the applications run long enough now to experience a soft fault.
Hardware Soft Errors

- In SRAMs, error correction codes (ECC) are generally hard-wired.
- Similarly, the communication interface to the cores is expected to have ECC for correcting errors that happen in communication over the on-chip network.
- The concern is about functional units:
  - soft errors in flip-flops, latches and
  - combinational logic (logic soft errors)
- The above soft errors are the most challenging...
- Built-in resilience is needed and must be provided in an optimal way combining hardware and software so that performance, power consumption efficiency, chip area etc are not sacrificed.
Challenges and opportunities

• Challenges
  – The challenges are to maintain robust performance in terms of performance, power consumption, chip area utilization, and design validation costs.
  – Classical redundancy assumptions do not hold because of density of components on the chip. No longer faults occur in a single component nor independent of each other.

• Opportunities
  – Hardware: long interconnects allow addition of local transistors and network without affecting chip area
  – Middleware (?) : High density non-volatile storage elements.
  – Multicore architecture supporting software enhancements for fault tolerance at algorithm level, e.g. check sums computed on a parity thread tuned to a particular core.
Distribution of Transient Faults

- Chips with high rate of transient faults get detected immediately after manufacture and would get rejected.
- Again, during aging more transient faults are likely.
- We focus our attention on the Normal Lifetime of the multi-core processor.
Reducing Transient Faults in Normal Life

- Usually circuit failure prediction based on-line self-test techniques are employed for early life and aging (negative bias temperature instability) troubles.

- During the normal life operations phase a combination of hardware and software based remedies is desirable
  - Hardware Based Soft Error Protection
    - Built-in self test and scan based techniques
    - Built in Soft Error Resilience
  - Software Based Protection
    - Algorithm based fault tolerance
- Hardware technique alone or software alone may not be very useful in terms of power, chip area, validation cost and fault tolerance overheads in the software library.
Hardware Based Soft Error Protection
source: Mitra 2008

- Hardened Latches and Flip-flops (Calin 1996)
  - Silent Data Corruption (SDC) and Detected but Uncorrected Errors (DUE) in Latches reduced 20 times, Combinational logic: none, power penalty ~18%, speed penalty ~1%, no die size increase, no extra validation; applies to all designs

- Hardware Duplication (Bartlett 2004)
  - almost all (both sequential and combinational elements) SDC protected but increased DUEs, energy penalty ~100%, small speed penalty, >40% die size increase, significant validation effort, configurable and applicable to all designs.

  - almost all SDC as above but increased DUEs. >40% energy penalty, significant validation effort, effective only for custom designs; die size is unclear.
BISER protection for Latches and Flip-flops (Mitra 2008)

BISER protection for Combinational Logic (Mitra 2008)
Hardware Soft Error Protection (BISER)

- BISER (built in soft error resilience) (Mitra 2008, 2006 and Zhang 2006) is a more recent and likely optimal effort.
  - Uses a BISER (redundant) Latch with scan reuse, a “C” element, and a weak keeper.
  - BISER reduces SDC on latches 20 times and on combinational logic by 12-64 times. DUE reduction is same as SDC.
  - Energy penalty is ~10%
  - Speed penalty is ~1% in sequential and ~5% in combinational elements
  - No significant extra real estate on the chip is needed.
  - No extra design validation efforts.
  - Configurable and applicable to all designs.
Optimal Soft Error Protection

- However, BISER cannot alone give all the soft error protection.
  - Energy penalty may peak to significantly high levels in the resilient mode if almost all error are to be protected against.
  - Due to ever denser design, the soft error probability rises and even when operating at full 20 times reduction capacity, the probability may still be significant.
  - Especially for scientific applications that run for a long time BISER cannot protect against all soft errors.

- BISER must be combined with linear algebra specific algorithm based fault tolerance. (Mitra 2006, 2007, 2008)

- Check pointing with stop and restart is not an option given the significant probability of soft errors and the $O(n^2 \lg n)$ complexity of the threaded linear algebra routines.
Algorithm Based Fault Tolerance (ABFT) for Numerical Linear Algebra

- Like BISER, using ABFT alone would be cumulatively and computationally too costly for error protection. Hence these must be combined.

- ABFT (Huang 1984, Anfinson & Luk 1988) for linear algebra was originally proposed in the 1980's to work with early but yet to be reliable hardware of large parallel machines.

- Objective is to provide low-cost real time error protection.

- For numerical linear algebra, ABFT uses analogous concepts in real two-dimensional arrays as ECC for binary data.
  - a distance (similar to Hamming distance) $d+1$ code can be used to detect up to $d$ errors and to correct up to $\lfloor d/2 \rfloor$ errors.
  - does not work for general weight vectors though.
Checksum Matrices

- Checksum approach of Huang & Abraham (mid-1980's) can detect a single soft (transient) error on an input $n \times n$ matrix. The row checksum matrix is $A_r := [A \ Ae]$ and the column checksum matrix is $A_c := [A^T \ A^T e]^T$ where $e = [1,1,1,...,1]^T$. By comparing $\sum_{j=1,n} a_{ij}$ with $(Ae)_i$ we can detect the error in the $i^{th}$ row. Similarly compare $\sum_{i=1,n} a_{ij}$ with $(e^T A)_j$ to detect the error in the $j^{th}$ column.

- Full $(n+1) \times (n+1)$ checksum matrix $A_f$ is given as

$$A_f := \begin{bmatrix} A & Ae \\ e^T A & e^T A e \end{bmatrix}$$

from which we determine the exact location of the error by intersecting inconsistent rows and columns. It can be used with only one row or column too.
Weighted Checksum approach

- Weighted checksums give increased error protection.

- With $d$ weighted checksum columns/rows for a suitable choice of weight vectors at most $d$ errors are detected and $\text{floor}(d/2)$ errors are corrected.

- For unique $n \times 1$ weight vectors $w^{(i)}$, $i=1,...,d$ with unique elements $w^{(i)}_j$, $j=1,...,n$ we define the $n \times (n+d)$ weighted row checksum matrix $A_{rw}$ as $[ A \ A w^{(1)} \ A w^{(2)} \ldots \ A w^{(d)}]$ and the $(n+d) \times n$ weighted column checksum matrix $A_{cw}$ as $[ A^T \ A^T w^{(1)} \ldots \ A^T w^{(d)}]^T$.

- For $d=2$, with $w^{(1)}=e$, $w^{(2)}=w$ assume that no errors (protected by BISER, for example) have occurred during the formation of checksum rows and that one error has occurred in $a_{pq}$ and the erroneous $a_{ij}$ is denoted as $\tilde{a}_{ij}$, $1 \leq i, j \leq n$. 
Weighted Checksums (continued)

- Consider the column weighted checksum matrix $A_{cw}$. Then form the sums $s_1 = \sum_{k=1}^{n} \bar{a}_{kq} - (Ae)_q$ and $s_2 = \sum_{k=1}^{n} w_k \bar{a}_{kq} - (Aw)_q$. Examine if $s_2 / s_1 = \omega_q$. We do similarly for the row weighted checksum. Then we have properly located the error in $(p,q)$ position of $A$. To correct it, simply replace $a_{pq}$ with $\bar{a}_{pq} - s_1$.

- In checking the checksum a tolerance level must be chosen to take account of rounding errors. The problem that plagues floating point checksums is the overflow due to relatively large weights. In practice, Jou and Abraham have used $\omega_j^{(i)} = 2^{(j-1)(i-1)}$; others have implemented with $\omega_j = j$. A possibility is implementing efficient hardware that uses modulo arithmetic for computing and checking checksums.
Definitions

- Similar to parity check matrix in coding theory, we define a matrix $H$ so that its null space (code space) defines the code $C$.

- A vector $x$ is said to be error-free if $Hx=0$, i.e., $x$ is in $null(H)$.

- A non-code vector lies in the domain of $H$ but not in $null(H)$.

- For weighted column (row) checksums, a vector $x$ is considered error-free if at the end of computations it satisfies $Hx=0$. That is, for example, if all the columns of $A_{cw}$ are code vectors then $A$ is error free.

- $s$ is syndrome vector and $H\tilde{y}=s$; $\tilde{y}$ is the possibly erroneous version of a vector $y$. Correction vector $c=\tilde{y}-y$.

- $dist(\tilde{y},y)$ between two vectors $\tilde{y}, y$ equals the number of entries which differ. This is a metric and is zero when $\tilde{y}, y$ coincide.
Definitions (continued)

- The \( \text{dist} \) satisfies triangular inequality and the weight of a vector \( v \) in the domain of \( H \) equals \( \text{dist}(v, 0) \).
- The distance of the code space \( C \) is the minimum of the distances between all possible pairs of nonzero vectors in \( \text{null}(H) \).
- The weight of code space \( C \) is equal to minimum weight of all nonzero vectors in \( \text{null}(H) \).
- The \( d \times (n+d) \) weighted checksum matrix \( H \) is

\[
H = \begin{bmatrix}
  w_1^{(1)} & w_2^{(1)} & \ldots & w_n^{(1)} & -1 & 0 & \ldots & 0 \\
  w_1^{(2)} & w_2^{(2)} & \ldots & w_n^{(2)} & 0 & -1 & \ldots & 0 \\
  \vdots & \vdots & \ddots & \vdots & 0 & 0 & \ldots & 0 \\
  w_1^{(d)} & w_2^{(d)} & \ldots & w_n^{(d)} & 0 & 0 & \ldots & -1
\end{bmatrix}
\]

**Example:**

\[
H = \begin{bmatrix}
  1 & 1 & 1 & -1 & 0 \\
  1 & 2 & 2^2 & 0 & -1
\end{bmatrix};
\]

\[
b_1 = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \end{bmatrix}^T
\]

\[
b_2 = \begin{bmatrix} 0 & 1 & 0 & 1 & 2 \end{bmatrix}^T
\]

\[
b_3 = \begin{bmatrix} 0 & 0 & 1 & 1 & 2^2 \end{bmatrix}^T
\]

are basis for CodeSpace \( C(\text{distance } 3) \)
Guarantee of Error Detection

- based on Theorem (Anfinson-Luk): Let $C$ be the code space of matrix $H$. For each code vector $v$ with $\text{dist}(v, 0) = d+1$ there are $d+1$ linearly independent columns of $H$. Likewise, if there exists $d+1$ linearly dependent columns of $H$, then there is a code vector of weight $d+1$.

- If no $d$ or fewer columns of $H$ are linearly dependent, then the code has minimum weight at least $d+1$.

- The distance of $C$ is equal to the smallest number of linearly dependent columns of $H$.

- Let $l = \text{weight}(v)$. We see that if the matrix $H$ has every set of $d$ columns linearly independent, then we can detect up to $d$ errors because $Hv \neq 0$ whenever $l \leq d$. If $l > d$, then we can find some vector $v$ such that $Hv = 0$ since any $d+1$ or more columns of $H$ are linearly dependent. Thus no guarantee of error detection for $l > d$. 
Uniqueness of Correction

- If $H$ has every set of $d$ linearly independent columns, then we can detect at most $d$ errors. Given $l \leq d$. Then, there exists a $\nu$ such that $H\nu = s$ where $s$ is a syndrome vector and belongs to range of $H$. Thus a correction exists.

- The correction vector $\nu$ is unique when $l \leq \floor(d/2)$. If $\nu$ was not unique, then another vector $\nu_1$ would satisfy $H\nu_1 = s$. Thus $H(\nu_1 - \nu) = 0$; $(\nu_1 - \nu)$ is in $\text{null}(H)$. Hence $(\nu_1 - \nu)$ has at least $d+1$ nonzeros. Since $l \leq \floor(d/2)$ both $\nu$, $\nu_1$ can have at most $\floor(d/2)$ nonzeros. Then $(\nu_1 - \nu)$ can have at most $2*\floor(d/2) < d+1$ nonzeros. This is a contradiction and $\nu$ must be unique.

- The syndrome in our example is $[1 1]^T$. 
Finding the Correction Vector

Take \( d = 4 \). \( A_{cw} := \begin{bmatrix} A^T & A^T w_1 & A^T w_2 & A^T w_3 & A^T w_4 \end{bmatrix}^T \)

can correct 2 errors. For correcting 2 errors in column \( j \) of \( A_{cw} \).

Recall that \( v_i = \overline{a}_{ij} - a_{ij}, \ i = 1, 2, \ldots, n + 4 \) so that we must solve
\[
 w_k^{(m)} v_k + w_l^{(m)} v_l = s_m; \ m = 1, 2, 3, 4.
\]

This means 10 unknowns from 4 non-linear equations. For solving, need to have special relationship among \( w_j^{(i)} \)'s that still must be distinct.

Adding additional checksum columns do not help but merely increases two unknowns for each column.
Example

\[
H = \begin{bmatrix}
1 & 1 & 1 & -1 & 0 \\
1 & 2 & 2^2 & 0 & -1
\end{bmatrix};
\]

\[
b_1 = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \end{bmatrix}^T
\]

\[
b_2 = \begin{bmatrix} 0 & 1 & 0 & 1 & 2 \end{bmatrix}^T
\]

\[
b_3 = \begin{bmatrix} 0 & 0 & 1 & 1 & 2^2 \end{bmatrix}^T
\]

distance of code space is 3

\[
s = \begin{bmatrix} 1 & 1 \end{bmatrix}^T
\]

Case 1: \( l = 1 \); \( v = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix}^T \)

Case 2: \( l = 2 \); \( v \) is not unique.

\[
v = \begin{bmatrix} 0 & 3/2 & -1/2 & 0 & 0 \end{bmatrix}^T; \quad v = \begin{bmatrix} 0 & 0 & 1/4 & -3/4 & 0 \end{bmatrix}^T
\]

Case 3: \( l = 3 > 2 = d \). Let \( v = \begin{bmatrix} 2 & -3 & 1 & 0 & 0 \end{bmatrix}^T \). \( Hv = 0 \).

Error not detected
**Distance $d$ of code $C$ for particular weights**

Consider $w_j^{(i)} = 2^{(i-1)(j-1)}$ for $j=1,2,...,n$; $i=1,2,...,d$.

\[
\begin{bmatrix}
1 & 1 & \ldots & 1 & -1 & 0 & \ldots & 0 \\
1 & 2 & \ldots & 2^{n-1} & 0 & -1 & \ldots & 0 \\
1 & 2^2 & \ldots & 2^{2(n-1)} & 0 & 0 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & 2^{d-1} & \ldots & 2^{(d-1)(n-1)} & 0 & 0 & \ldots & -1
\end{bmatrix}
\]

Let $H_d$ be any $d$ columns of $H$. $D_d := \det(H_d)$.

$d$ of first $n$ columns of $H$ means $H_d$ is Vandermonde Matrix.

\[
\det(\text{Vandermonde}(w_1, w_2, \ldots, w_n)) = \prod_{1 \leq j \leq n} (w_i - w_j) \neq 0
\]

when weights are distinct.

Choose $H_d = \begin{bmatrix}
P & 0 \\
Q & -I_{d-i}
\end{bmatrix}$; $P_{ij} = \lambda^k_j$ form. $\det(P^T) = \det(P) \neq 0$.

$H_d$'s $i$ columns are from first $n$ columns

d$-i$ columns from $n+1$ thru $n+d$ columns.
Example Algorithm for Distance 5

- For $d=4$ (distance=$d+1=5$) case we need to solve
  \[ 2^{(i-1)k} v_k + 2^{(i-1)l} v_l = s_i , \ i=1,2,3,4. \]

  Substitute $\mu=2^k$ and $\psi=2^l$ in the above equations. Then compute the product $\eta=(s_1 s_4 - s_2 s_3)/(s_1 s_3 - s_2 s_2)$ with values of $s_i$ so that $\eta=\mu+\psi>0$. If $k>l$, then $2^k < \eta < 2^{k+1}$. Since $\eta$ and $k$ are integers we must have $k = \text{floor}(\log \eta)$ and compute $l=\log(\eta-2^k)$.

- It may be noticed that the correction vector depends upon how the weights are related.

- Thus error correction commonly known for GF(2) is extended for correcting errors in $n$ dimensional real space.
Implementation

- The error detection and correction discussed so far can be applied to various Basic Linear Algebra Subroutines (BLAS) operations.

- A thread or threads may be included in BLAS routines which would perform the error detection and correction.

- The check sum formation is to be protected by BISER like hardware based transient error reduction. This allows BISER to be used optimally without significant energy and performance penalty.

- A tuning issue will be the choice of optimum distance of the code space based on cache size, stride and data distribution size among the threads (cores).

- For an application using particular Linear Algebra routines (LAPACK) routines intensively, higher level ABFT (error protected LAPACK routines) may be prove more optimal.
Application to LAPACK routines

- We may apply the ABFT scheme to some of the LAPACK routines readily.
- However, for many LAPACK routines applying ABFT at this slightly higher level than BLAS remains non-trivial and needs to be investigated.
- Especially, ABFT at the level of Eigen-decomposition and Singular Value Decomposition remains open topics.
- Here we demonstrate application of the ABFT for real matrices to an important decomposition in numerical linear algebra (including LAPACK) libraries, viz., QR Decomposition which reveals the condition number of a matrix and computes the basis vector for its null space.
- In particular, we study Givens rotation based QR.
Givens Rotations

Givens zeros components selectively. These are Rank 2, orthogonal corrections to Identity.

\[
G(i, k, \theta) = \begin{bmatrix}
1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\
\vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\
0 & \cdots & \cos(\theta) & \cdots & \sin(\theta) & \cdots & 0 \\
\vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\
0 & \cdots & -\sin(\theta) & \cdots & \cos(\theta) & \cdots & 0 \\
\vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\
0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \\
\end{bmatrix}_{i \ k}
\]

Premultiplication by \(G(i, k, \theta)^T \rightarrow \text{CCW}\) rotation of \(\theta^c\) in \((i, k)\) coordinate plane

if \(x \in \mathbb{R}^n\), \(y = G(i, k, \theta)^T x\), then

\[
\begin{align*}
    y_j &= \begin{cases}
        cx_i - sx_k, & j = i \\
        sx_i + cx_k, & j = k \\
        x_j, & j \neq i, k
    \end{cases}
\end{align*}
\]

\(y_k\) can be zero by setting

\[
c = x_i / \sqrt{x_i^2 + x_k^2}, \quad s = -x_k / \sqrt{x_i^2 + x_k^2}
\]

\(function:\) \([c, s] = \text{givens}(a, b)\)

\(if\) \(b = 0\)

\(c = 1; s = 0\)

\(else\)

\(if\) \(|b| > |a|\)

\(\tau = -a/b; s = 1/\sqrt{1 + \tau^2}; c = s \tau\)

\(else\)

\(\tau = -b/a; c = 1/\sqrt{1 + \tau^2}; s = c \tau\)

\(end\)

\(end\)

\(\Rightarrow\) Given scalars \(a, b\) compute \(c, s\) s.t.

\[
\begin{bmatrix} c & s \end{bmatrix}^T \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}
\]

Requires 5 flops plus 1 square root.
No trigonometric functions are involved.
Applying Givens Rotation

\[ A \in \mathbb{R}^{m \times n}; G(i, k, \theta) \in \mathbb{R}^{m \times m}; A \leftarrow G(i, k, \theta)^T A \] implies \[ A(i, k, :) = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} A(i, k, :) \]

Requires 6n flops:

\[ \text{for } j = 1 : n \]
\[ \tau_1 = A(i, j) \]
\[ \tau_2 = A(k, j) \]
\[ A(1, j) = c \tau_1 - s \tau_2 \]
\[ A(2, j) = s \tau_1 + c \tau_2 \]

end

Update \[ A \leftarrow A G(i, k, \theta), \ G \in \mathbb{R}^{n \times n}, \ 6m \text{ flops} \]

\[ A(\cdot, [i, k]) = A(\cdot, [i, k]) \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \]

\[ \text{for } j = 1 : m \]
\[ \tau_1 = A(j, i) \]
\[ \tau_2 = A(j, k) \]
\[ A(j, i) = c \tau_1 - s \tau_2 \]
\[ A(j, k) = s \tau_1 + c \tau_2 \]

end

RoundOff Properties:

\[ \hat{c} = c(1 + \epsilon_c) \quad \epsilon_c = O(u) \]
\[ \hat{s} = s(1 + \epsilon_s) \quad \epsilon_s = O(u) \]
\[ \text{fl}(\hat{G}^T A) = G^T (A + E) \quad \|E\|_2 \approx u \|A\|_2 \]
\[ \text{fl}(A \hat{G}) = (A + E) G \quad \|E\|_2 \approx u \|A\|_2 \]

Let \[ Q = G_1 \ldots G_t \]. Similar to Householder being economical in factored form than explicit associate single floating point number \( \rho \) with each rotation:

\[ Z = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \quad c^2 + s^2 = 1 \]

Define \( \rho \): if \( c = 0, \rho = 1 \) elseif \( |s| < |c|, \rho = \text{sign}(c) s / 2 \) else \( \rho = 2 \text{sign}(s) / c \) end
**Givens QR**

\[
\begin{bmatrix}
\times & \times & \times \\
\times & \times & \times \\
\times & \times & \times \\
\end{bmatrix} \rightarrow \begin{bmatrix}
\times & \times & \times \\
\times & \times & \times \\
0 & \times & \times \\
\end{bmatrix} \rightarrow \begin{bmatrix}
\times & \times & \times \\
0 & \times & \times \\
0 & \times & \times \\
\end{bmatrix} \rightarrow \begin{bmatrix}
\times & \times & \times \\
0 & 0 & \times \\
0 & 0 & \times \\
\end{bmatrix} \rightarrow R
\]

\(G_j\) is \(j^{th}\) Givens Rotation: \(Q^T A = R\) upper triangular; \(Q = G_1...G_t\), \(t = \text{total no. rot.}\)

**Givens QR.** \(A \in \mathbb{R}^{m \times n}, m \geq n\), overwrites \(A\) with \(Q^T A = R\), \(Q\) orthogonal.

for \(j = 1 : n\)

\[
\begin{align*}
\text{for } i &= m : -1 : j + 1 \\
[c, s] &= \text{givens} (A(i - 1, j), A(i, j)) \\
A(i - 1 : i, j : n) &= \begin{bmatrix} c & s \\ -s & c \end{bmatrix} A(i - 1 : i, j : n)
\end{align*}
\]

end

end

Requires \(3n^2(m - n/3)\) flops. Use \(\rho\) single number encoding of \((c, s)\) stored in zeroed \(A(i, j)\). \(x \leftarrow Q^T x\) is implemented, with rotations reconstructed in proper order. Different sequences of rotations such as

for \(i = m : -1 : 2\)

\[\text{for } j = 1 : \min(i - 1, n)\]

zeroes \(A\) row-by-row.

Different planes of rotation for zeroing \(a_{ij}\)

Using rows \(j, i\) instead of \(i - 1, i\)

\[
[c, s] = \text{givens} (A(j, j), A(i, j)) \\
A([j \ i], j : n) = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} A([j \ i], j : n)
\]
Transient Error affecting Givens QR

- Assume that a single transient error may appear in the \( i \)-th iterate.

- (Maslennikow et. al.) If at the \( i \)-th step, an element \( \alpha_{jk}^{(i+1)} \) (\( i < j, i < k \)) is wrongly calculated, then the errors will not appear among other elements of \( A^{(i+1)} \). The parentehsized superscript refers to iterate number.

- However, if \( \alpha_{jk}^{(i)} \) is erroneous, then error appears while computing either the element \( \alpha_{jk}^{(i+1)} \) in the pivot row at the \( j \)-th step of the algorithm, for \( j \leq k \), or in the values of \( \alpha_{kk}^{(j)}, c \) and \( s \) (\( j = i+1 \) to \( M \)) at the \( k \)-th step, for \( j > k \). Hence in these cases, we should check and possibly correct elements of the \( i \)-th and \( j \)-th rows of \( A^{(i)} \), each time after their recomputing.
Transient Error Affecting Givens QR (contd)

(Maslennikow et. al.) Let an element $a^{(j)}_{ik}$ of the pivot row $(j = i + l, \ldots, M; k = 1, \ldots, N)$ or an element $a^{(i+1)}_{jk}$ of a non-pivot row $(k = i + 1, \ldots, N)$ was wrongly calculated when applying the Givens rotation. Then it is possible to correct its value using the Weighted Checksum method for the row encoded matrix $A_r = [A \ A_p \ A_q]$.

To detect errors, one can check $c^2 + s^2 = 1$.

also check if $\|A\|_2$ is preserved during computation.

Thus we can have triple redundancy in checks.

The checks and checksum computations are assumed to be BISER protected.
ABFT Givens QR

1. The original matrix $A$ ($M \times N$) is represented as the row encoded matrix.

2. For $i = 1, 2, \ldots, K$, stages 3-9 are repeated.

3. The values of $a_{j}^{(j)}$ are calculated, $j = i + 1, \ldots, M$.

4. The 2-norm for the $i$-th column of $A$ is calculated. This stage needs approximately $M-i$ multiply-add operations. The value of $||a_i||$ is compared with the value of $||a_i||$ (present iterate). If not equal, then stages 3,4 are repeated.

5. The coefficients $c$ and $s$ are computed and checked for $c^2 + s^2 = 1$. In case of non-equality, stage 5 is repeated.
6. For \( j = i + 1, \ldots, M \), stages 7-10 are repeated.

7. The elements \( a_{ik}^{(j)} \) of the \( i \)-th row of \( A^{(i+1)} \) are computed for \( k = 1, \ldots, N + 1 \).

8. \((A^{(j)p})_i, (A^{(j)q})_i\) are computed. This stage needs approximately \( N - i \) additions. If error is detected, correction is computed. Stages 7,8 are performed again if needed.

9. The elements \( a_{jk}^{(i+1)} \) are calculated for \( k = i+1, \ldots, N+1 \).

10. \((A^{(i+1)p})_j, (A^{(i+1)q})_j\) are computed. If error is detected, correction is computed. If needed stages 9,10 are repeated.

Extra Work: \( O(4.5*N*N) \). Corrects \( N*N \) single errors in whole Givens QR process. Scales as \( N*N*N/(\#\text{cores}) \) when threaded.
Research Issues

- Distance of the code space for an energy optimal robust performance of the processor.
- Tuning of the ABFT part of the computation to BISER protected functional units of the processor.
- BISER-ABFT interaction in terms of energy consumption, performance degradation etc.
- New speed up and efficiency metrics need to be defined accordingly.
- ABFT for higher level LAPACK routines: SVD, Eigenvalues etc

Need: BISER/BIST protected multi-core hardware with Fault Injection facility for proper simulation of ABFT BLAS and LAPACK routines.
References and Sources


- Mitra, S. , Globally Optimized Robust Systems to Overcome Scaled CMOS Reliability Challenges, Design Automation and Test in Europe, January, 2008 - gigascale.org

Thank You!