

20 Years of Innovative Computing
Knoxville, Tennessee
March 26th, 2010

Fault Tolerate Linear Algebra:
Survive Fail-Stop Failures without Checkpointing

Zizhong (Jeffrey) Chen
zchen@mines.edu

Colorado School Mines

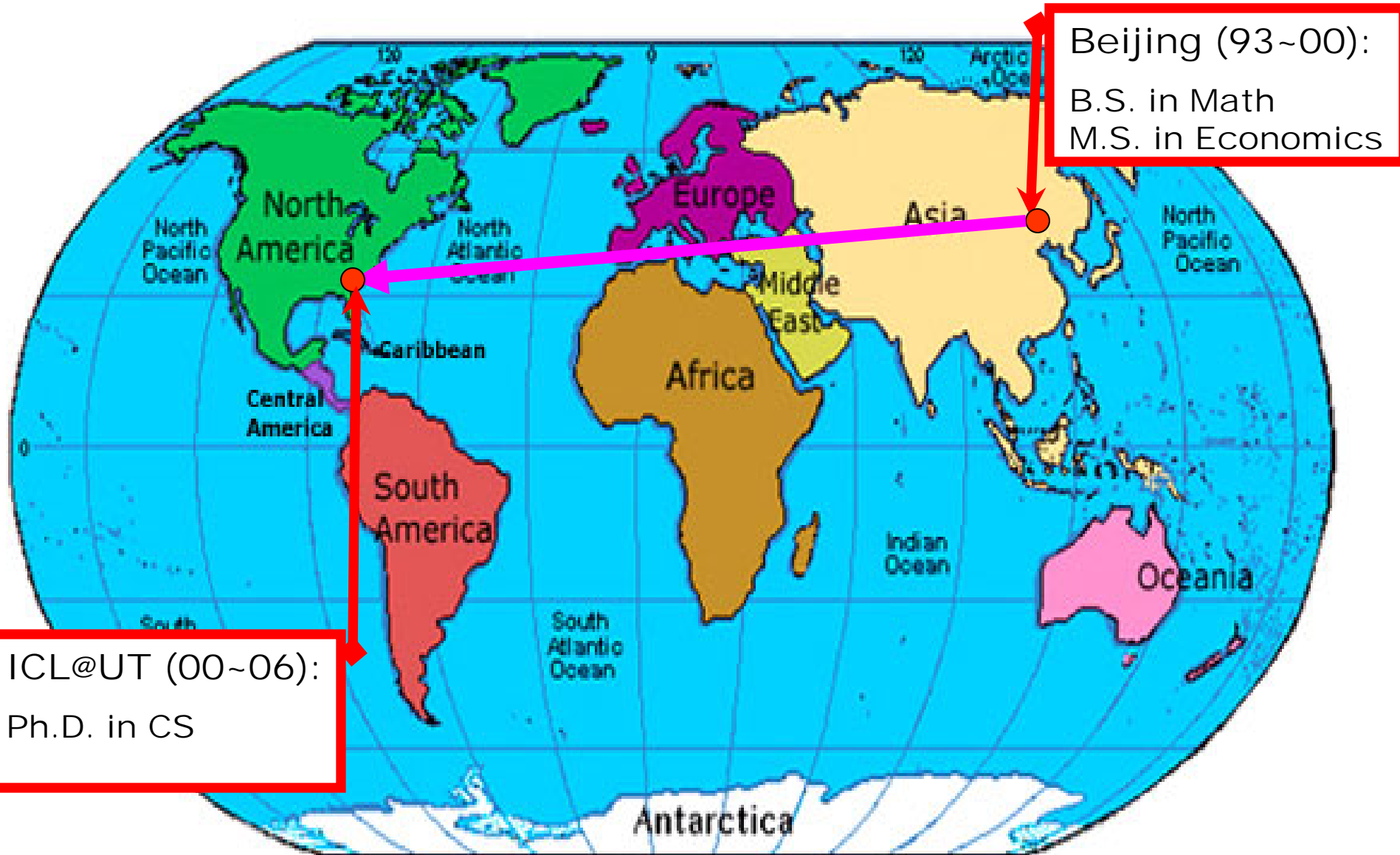
My Past



Beijing (93~00):
B.S. in Math
M.S. in Economics

Chong-Qing:
Education before
universities

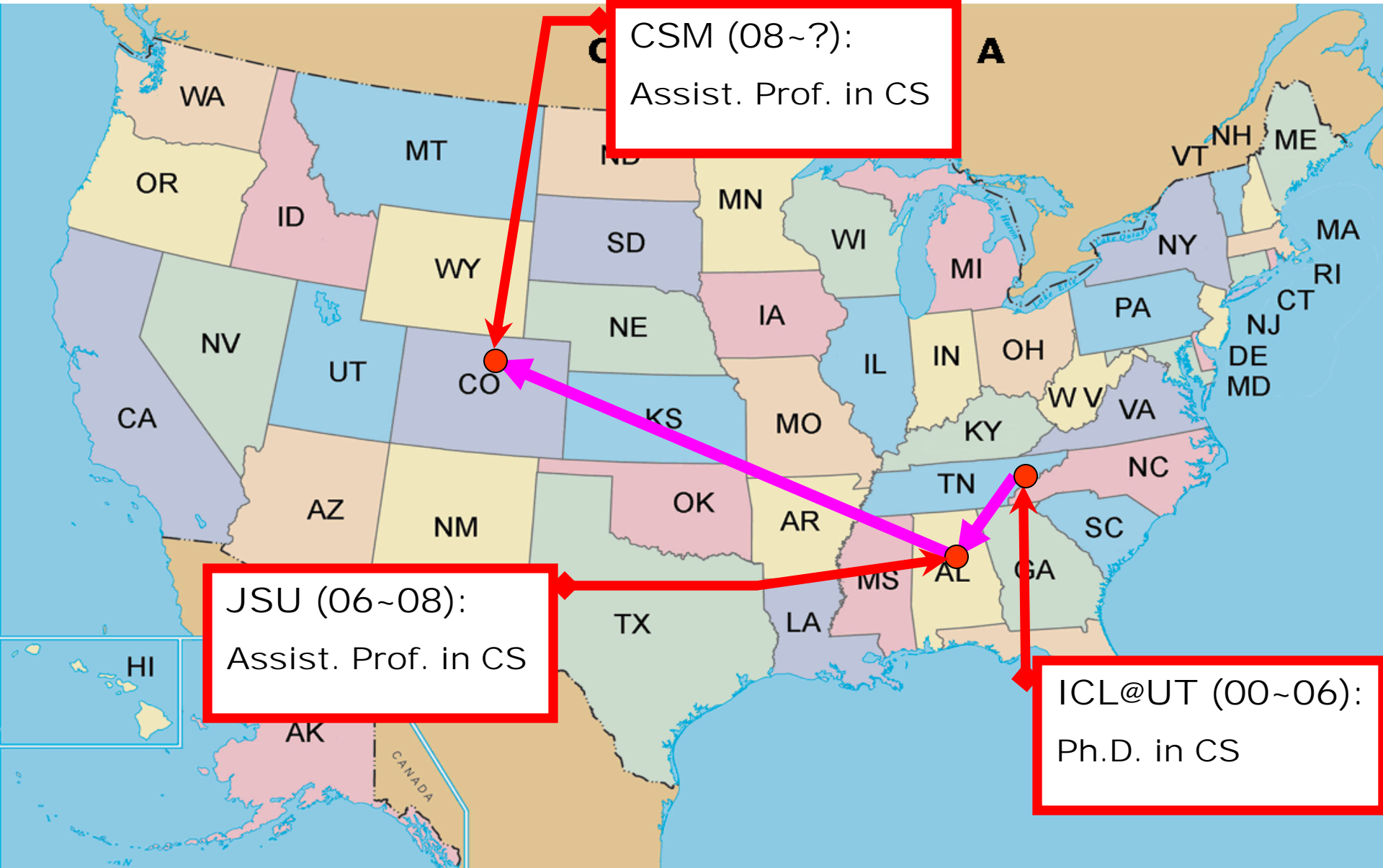
My Past



Beijing (93~00):
B.S. in Math
M.S. in Economics

ICL@UT (00~06):
Ph.D. in CS

My Past



CSM (08~?):
Assist. Prof. in CS

JSU (06~08):
Assist. Prof. in CS

ICL@UT (00~06):
Ph.D. in CS

Outline

- **Motivation**
- **Expected Program Execution Time**
- **Fault Tolerant Linear Algebra**
- **Floating-point Error/Erasure Correcting Codes**

Reliability of Large Systems

- **Parallel Systems with thousands of cores very common today**
- **Failure of some links or cores is becoming more and more frequent**

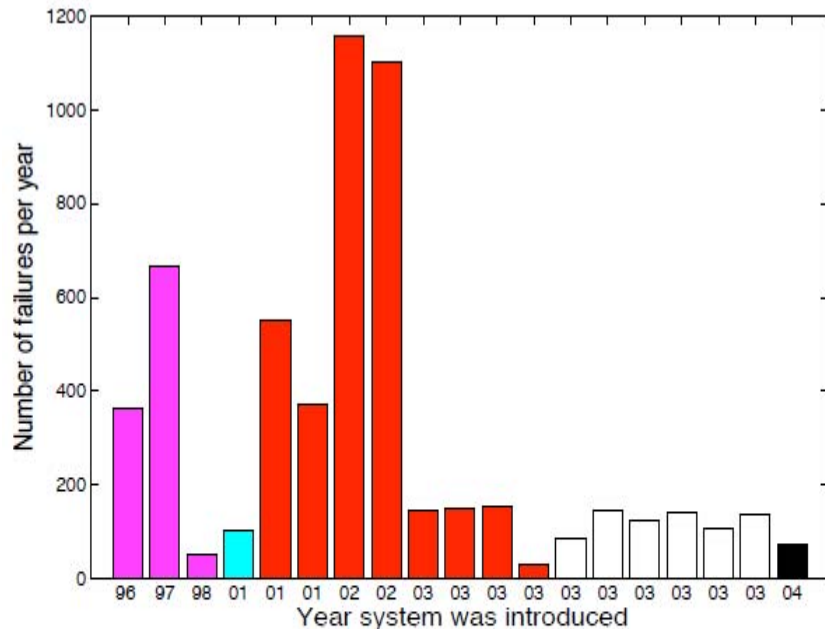
Wiring of interconnection network cables



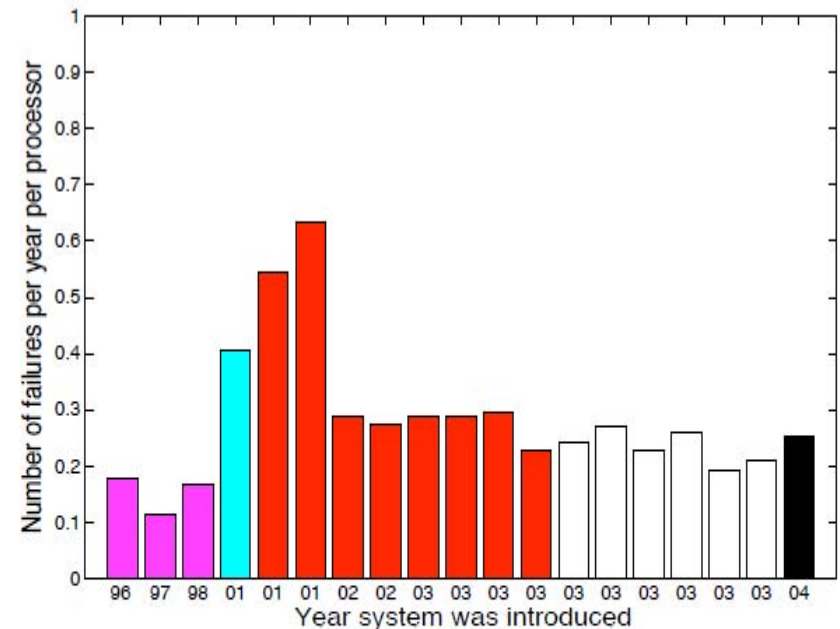
Failure Rate in HPC Practice

Gibson (CMU): Fault Tolerance in Petascale Computers, CTWatchQuarterly, Nov, 2007

MTBF: from about half a month to less than half a day!



(a)



(b)

Figure 2. (a) Average number of failures for each LANL system per year. (b) Average number of failures for each system per year normalized by number of processors in the system. Systems with the same hardware type have the same color.

Expected Program Execution Time without Fault Tolerance

- Suppose the MTBF for a single processor is M
 - The failure rate of a single processor is $\lambda = 1 / M$
- What is the failure rate of p processors with i.i.d. $\exp(\lambda)$?
 - $p\lambda$
- Expected program execution time
 - T : program execution time in a failure free environment
 - E_{non-ft} : expected program execution time in an unstable environment

$$\begin{aligned} E_{non-ft} &= (e^{p\lambda T} - 1) \frac{1}{p\lambda} \\ &= (e^{p \frac{T}{M}} - 1) \frac{M}{p} \end{aligned}$$

Expected Program Execution Time without Fault Tolerance

- Suppose the MTBF for a single processor is M
 - The failure rate of a single processor is $\lambda=1/M$
- What is the failure rate of p nodes with i.i.d. $\exp(\lambda)$?
 - $p\lambda$
- Expected program execution time
 - T : program execution time in a failure free environment
 - E_{non-ft} : expected program execution time in an unstable environment

$$\begin{aligned} E_{non-ft} &= (e^{p\lambda T} - 1) \frac{1}{p\lambda} \\ &= (e^{p \frac{T}{M}} - 1) \frac{M}{p} \end{aligned}$$

Expected Program Execution Time with Checkpointing/Rollback

- $t_c(p)$: dedicated time for one checkpoint
- $t_r(p)$: dedicated time for one recovery
- $E_{checkpoint-opt}$: expected program execution time in an unstable environment with optimal checkpoint interval

$$E_{checkpoint-opt} \approx T \left(1 + \frac{t_r(p)}{\frac{M}{p}} \right) \frac{1}{1 - \sqrt{\frac{2t_c(p)}{\frac{M}{p}}}}$$

Expected Program Execution Time with Checkpointing/Rollback

- $t_c(p)$: dedicated time for one checkpoint
- $t_r(p)$: dedicated time for one recovery
- $E_{checkpoint-opt}$: expected program execution time in an unstable environment with optimal checkpoint interval

$$E_{checkpoint-opt} \approx T \left(1 + \frac{t_r(p)}{\frac{M}{p}} \right) \frac{1}{1 - \sqrt{\frac{2t_c(p)}{\frac{M}{p}}}}$$

increase quickly

Expected Program Execution Time with Checkpointing/Rollback

- $t_c(p)$: dedicated time for one checkpoint
- $t_r(p)$: dedicated time for one recovery
- $E_{checkpoint-opt}$: expected program execution time in an unstable environment with optimal checkpoint interval

$$E_{checkpoint-opt} \approx T \left(1 + \frac{t_r(p)}{\frac{M}{p}} \right) \frac{1}{1 - \sqrt{\frac{2t_c(p)}{\frac{M}{p}}}}$$

increase quickly =? decrease quickly

Expected Program Execution Time with Checkpoint-Free Recovery

- $t_{rr}(p)$: dedicated time for one recovery
- $E_{checkpoint-free}$: expected program execution time in an unstable environment with optimal checkpoint interval

$$E_{checkpoint-free} \approx T \left(1 + \frac{t_{rr}(p)}{\frac{M}{p}} \right)$$

Expected Program Execution Time with Checkpoint-Free Recovery

- $t_{rr}(p)$: dedicated time for one recovery
- $E_{checkpoint-free}$: expected program execution time in an unstable environment with optimal checkpoint interval

$$E_{checkpoint-free} \approx T \left(1 + \frac{t_{rr}(p)}{\frac{M}{p}} \right)$$

$$\begin{aligned} R_{overhead-ckpt-alg} &= \frac{E_{checkpoint} - E_{checkpoint-free}}{E_{checkpoint-free}} \\ &\approx \frac{1}{1 - \sqrt{\frac{2t_c(p)}{\frac{M}{p}}}} - 1 \end{aligned}$$

Fault Tolerant Linear Algebra: Survive Failures w/t Checkpointing

- Assume
 - $P_1(t) + P_2(t) + \dots + P_n(t) = P_{n+1}(t)$ for any t
 - $P_i(t)$: the data on the i^{th} processor at time t
- If the first processor fails, how can we recover the lost data $P_1(t)$?
 - Answer: $P_1(t) = P_{n+1}(t) - P_2(t) - \dots - P_n(t)$
- **Question 1:** does this kind of special relationship exist in non-trivial applications ?
 - YES, in many iterative methods, it does exist !
- **Question 2:** If such a relationship does not exist, is it possible for us create this kind of special relationship on purpose ?
 - YES, for many dense linear operations, such relationship can be created by performing computation with checksum matrices

Iterative Methods: Survive Failures w/t Checkpointing

- **Assume**
 - we are solving $A * x = b$ by iterative methods
 - $r = b - A * x$ is maintained in memory
- In a parallel environment, $r = b - A * x$ can be rewrite as

$$\begin{cases} a_{11} * x_1 + \dots + a_{1n} * x_n = b_1 - r_1 \\ \vdots \\ a_{n1} * x_1 + \dots + a_{nn} * x_n = b_n - r_n \end{cases}$$

- If the i^{th} processor fails, then both r_i and x_i become unknown
 - x_i can be recovered accurately unless a_{ki} is singular for ALL $k \neq i$
 - If a_{ii} is the only non-singular, set $r_i = 0$

Dense Linear Algebra: Survive Failures w/t Checkpointing

The *column checksum* matrix A^c of the matrix A is defined by

$$A^c = \begin{pmatrix} a_{00} & \cdots & a_{0n-1} \\ \vdots & \cdots & \vdots \\ a_{m-10} & \cdots & a_{m-1n-1} \\ \sum_{i=0}^{m-1} a_{i0} & \cdots & \sum_{i=0}^{m-1} a_{in-1} \end{pmatrix};$$

Algorithm-based Fault Tolerance
Abraham (UTexas), 1984:

$$\begin{array}{l} \text{If} \\ \text{Then} \end{array} \quad \begin{array}{l} A * B = C \\ A^c * B^r = C^f \end{array}$$

The *row checksum* matrix A^r of the matrix A is defined by

$$A^r = \begin{pmatrix} a_{00} & \cdots & a_{0n-1} & \sum_{j=0}^{n-1} a_{0j} \\ \vdots & \cdots & \vdots & \vdots \\ a_{m-10} & \cdots & a_{m-1n-1} & \sum_{j=0}^{n-1} a_{m-1j} \end{pmatrix};$$

Checksum is maintained in
final computation results

The *full checksum* matrix A^f of the matrix A is defined by

$$A^f = \begin{pmatrix} a_{00} & \cdots & a_{0n} & \sum_{j=0}^{n-1} a_{0j} \\ \vdots & \cdots & \vdots & \vdots \\ a_{m-10} & \cdots & a_{m-1n-1} & \sum_{j=0}^{n-1} a_{m-1j} \\ \sum_{i=0}^{m-1} a_{i0} & \cdots & \sum_{i=0}^{m-1} a_{in-1} & \sum_{j=0}^{n-1} \sum_{i=0}^{m-1} a_{ij} \end{pmatrix}.$$

Is the Checksum Maintained **During** Computation?

```
/* Calculate  $A^c * B^r$  by cannon's algorithm. */
initialize  $C = (c_{ij}) = 0$ ;
for  $i = 0$  to  $n - 1$ 
    left-circular-shift row  $i$  of  $A^c$  by  $i$ 
    so that  $a_{i,j}$  is overwritten by  $a_{i, (j+i) \bmod n}$ ;
end
for  $i = 0$  to  $n - 1$ 
    up-circular-shift column  $i$  of  $B^r$  by  $i$ 
    so that  $b_{i,j}$  is overwritten by  $b_{(i+j) \bmod n, j}$ ;
end
for  $s = 0$  to  $n - 1$ 
    every processor  $(i, j)$  performs  $c_{ij} = c_{ij} + a_{is} * b_{sj}$ 
    locally in parallel;
    left-circular-shift each row of  $A^c$  by 1;
    up-circular-shift each column of  $B^r$  by 1;
    /* Here is the end of the  $s^{th}$  step. */
end
```

```
/* Calculate  $A^c * B^r$  by fox's algorithm. */
initialize  $C = (c_{ij}) = 0$ ;
for  $s = 0$  to  $n - 1$ 
    for  $i = 0$  to  $n - 1$  in parallel
        processor  $(i, (i + s) \bmod n)$  broadcast local
             $t = a_{i, (i+s) \bmod n}$  to other processors in row  $i$ ;
    for  $i, j = 0$  to  $n - 1$  in parallel
        every processor  $(i, j)$ 
            performs  $c_{ij} = c_{ij} + t * b_{ij}$  locally;
    up-circular-shift each column of  $B^r$  by 1;
    /* Here is the end of the  $s^{th}$  step. */
end
```

```
/* Calculate  $A_r * B_c$  by outer product algorithm.*/
initialize  $C = (c_{ij}) = 0$ ;
for  $s = 0$  to  $n - 1$ 
    row broadcast the  $s^{th}$  row of  $A_c$ ;
    column broadcast the  $s^{th}$  column of  $B_r$ ;
    every processor  $(i,j)$  performs  $c_{ij} = c_{ij} + a_{is} * b_{sj}$ 
    locally in parallel;
    /* Here is the end of the  $s^{th}$  step. */
end
```

NO

YES

- **NOT maintained**
 - **Cannon's algorithm**
 - **Fox's algorithm**
- **Maintained**
 - **Outer product version algorithm**
- **Outer product version is usually used in today's HPC practice**

ScaLAPACK: Checksum from Processor Point of View

- Define the *row distributed checksum* matrix of M as

$$M^r = \begin{pmatrix} M_{11} & \cdots & M_{1q} \\ \vdots & \cdots & \vdots \\ M_{p1} & \cdots & M_{pq} \\ \sum_{i=1}^p M_{i1} & \cdots & \sum_{i=1}^p M_{iq} \end{pmatrix}$$

- Define the *column distributed checksum* matrix of M as

$$M^c = \begin{pmatrix} M_{11} & \cdots & M_{1q} & \sum_{j=1}^q M_{1j} \\ \vdots & \cdots & \vdots & \vdots \\ M_{p1} & \cdots & M_{pq} & \sum_{j=1}^q M_{pj} \end{pmatrix}$$

- Define the *full distributed checksum* matrix of M as

$$M^f = \begin{pmatrix} M_{11} & \cdots & M_{1q} & \sum_{j=1}^q M_{1j} \\ \vdots & \cdots & \vdots & \vdots \\ M_{p1} & \cdots & M_{pq} & \sum_{j=1}^q M_{pj} \\ \sum_{i=1}^p M_{i1} & \cdots & \sum_{k=1}^p M_{ik} & \sum_{i=1}^p \sum_{j=1}^q M_{ij} \end{pmatrix}$$

Overhead and Scalability Analysis (PDGEMM)

- The overhead (%) for constructing Checksum at the beginning

$$\begin{aligned}R_{total_encode} &= \frac{T_{total_encode}}{T_{matrix_mult}} \\ &= O\left(\frac{1}{Pm}\right)\end{aligned}$$

- The overhead (%) for computation due to larger size of matrices

$$\begin{aligned}R_{overhead_matrix_mult} &= \frac{T_{overhead_matrix_mult}}{T_{matrix_mult}} \\ &= O\left(\frac{1}{Pm}\right)\end{aligned}$$

- The overhead (%) for recovery after a failure

$$\begin{aligned}R_{recover_data} &= \frac{T_{recover_data}}{T_{matrix_mult}} \\ &= O\left(\frac{1}{Pm}\right)\end{aligned}$$

- Note

$$- O\left(\frac{1}{p*m}\right) \longrightarrow 0, \text{ as } p, m \longrightarrow \infty$$

Status

- **Matrix-matrix multiplication: Done**
- **Cholesky: Done**
- **LU: Almost Done**
- **QR: In progress**
- **Iterative Methods: Done**

Multiple Failures: Floating-Point Error Correcting Codes

$$\left\{ \begin{array}{l} C_1 = a_{11} * P_1 + \dots + a_{1j} * P_j + \dots + a_{1(j+m)} * P_{j+m} + \dots + a_{1n} * P_n \\ \vdots \\ C_m = a_{m1} * P_1 + \dots + a_{mj} * P_j + \dots + a_{m(j+m)} * P_{j+m} + \dots + a_{mn} * P_n \end{array} \right.$$

- In order to be able to recover from any k ($k \leq m$) failures, the weight matrix A has to satisfy
 - **Any square sub-matrix of A is non-singular**
- To maintain the checksum relationship
 - **Floating-point arithmetic** has to be used when calculating encodings
- Due to round-off errors in floating-point computations
 - **Require any square sub-matrix of A is well-conditioned**
- Can we use Cauchy, Vantermonde, DFT matrices ? **No!**

Floating-Point Codes Based on Random Matrices

- Gaussian random matrix
 - Well-conditioned with high probability
 - Simple to generate
- n checksum processors, p comp processors, m ($\leq n$) processors failed

$$\frac{1}{\sqrt{2\pi}} \left(\frac{0.245 \frac{n}{|n-m|+1}}{x} \right)^{|n-m|+1} < \Pr(\kappa_2(G_{m \times n}) > x) < \frac{1}{\sqrt{2\pi}} \left(\frac{6.414 \frac{n}{|n-m|+1}}{x} \right)^{|n-m|+1}$$

$$E(\log_{10} \kappa_2(G_{m \times n})) < \log_{10} \frac{n}{|n-m|+1} + 0.981.$$

Real Number Codes from Grassmannian Frames

A finite frame $\{x_k\}_{k=1}^N \subseteq \mathbb{R}^d$ is characterized by the property that its span is \mathbb{R}^d

Definition 1. Let $N \geq d$ and let $X_d^N = \{x_k\}_{k=1}^N$ be a subset of \mathbb{R}^d with each $\|x_k\| = 1$. The *maximum correlation* of X_d^N , $\mathcal{M}_\infty(X_d^N)$, is defined as

$$\mathcal{M}_\infty(X_d^N) = \max_{k \neq l} |\langle x_k, x_l \rangle|.$$

Real Number Codes from Grassmannian Frames

Definition 2. Let $N \geq d$. A sequence $U_d^N = \{u_k\}_{k=1}^N \subseteq \mathbb{R}^d$ of unit-norm vectors is an (N, d) -Grassmannian frame if it is a frame and if

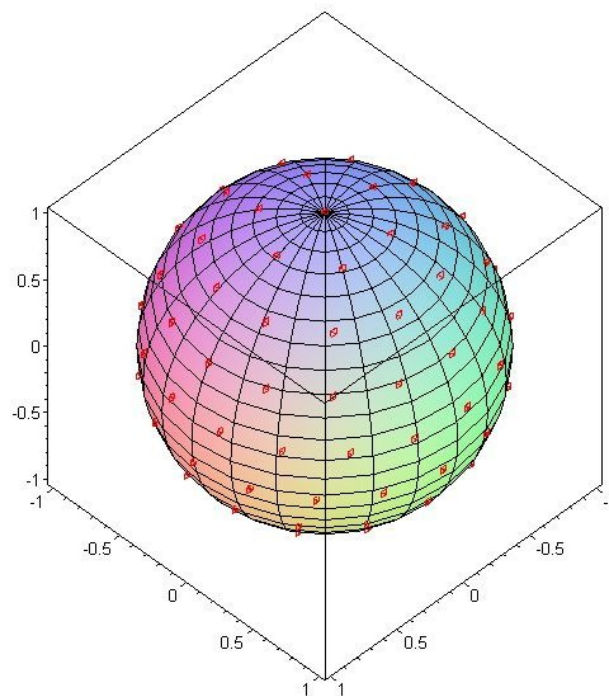
$$\mathcal{M}_\infty(U_d^N) = \inf \left\{ \mathcal{M}_\infty(X_d^N) \right\},$$

Distributing points on sphere appropriately
is usually difficult

**Stephen Smale @ Berkeley: 18 unsolved
math problems in 21st century**

**Question #7: distribution of points on
the 2-sphere**

**No analytical solution available except
for very few combination of N and d**



Best Line Packing in Grassmannian Space

- Minimizing the maximum correlation between pair vectors equals to maximize the minimum angle between pair of vectors
 - Packing lines in Grassmannian space
- Neil J. A. Sloane @ AT &T Research
 - Known optimal line packing for selected m , and n
 - For most m and n : it is unknown
 - Computational approximations
 - <http://www.research.att.com/~njas/grass/grassTab.html>
- Grassmannian Codes
 - Does NOT guarantee sub-matrices are well conditioned
 - It is even possible to contain singular sub-matrices

Real Number Codes with Optimal Numerical Stability

$$\left\{ \begin{array}{l} C_1 = g_{11} * P_1 + \dots + g_{1j} * P_{j+1} + \dots + g_{1(j+m)} * P_{j+m} + \dots + g_{1n} * P_n \\ \vdots \\ C_m = g_{m1} * P_1 + \dots + g_{mj} * P_{j+1} + \dots + g_{m(j+m)} * P_{j+m} + \dots + g_{mn} * P_n \end{array} \right.$$

- **Recovery accuracy**
 - NOT directly related to the *correlation of pair of vectors*
 - BUT more directly related to the *condition number* of the coefficient matrix
- **Real number codes with optimal numerical stability**
 - Optimize the recovery accuracy for the worst case scenario
 - Minimize maximum condition number of the coefficient matrix
 - **Defined optimal real number codes as G that satisfies**

$$f(m, n) = \min_G \max_i \kappa(G_i)$$

Real Number Codes with Optimal Numerical Stability

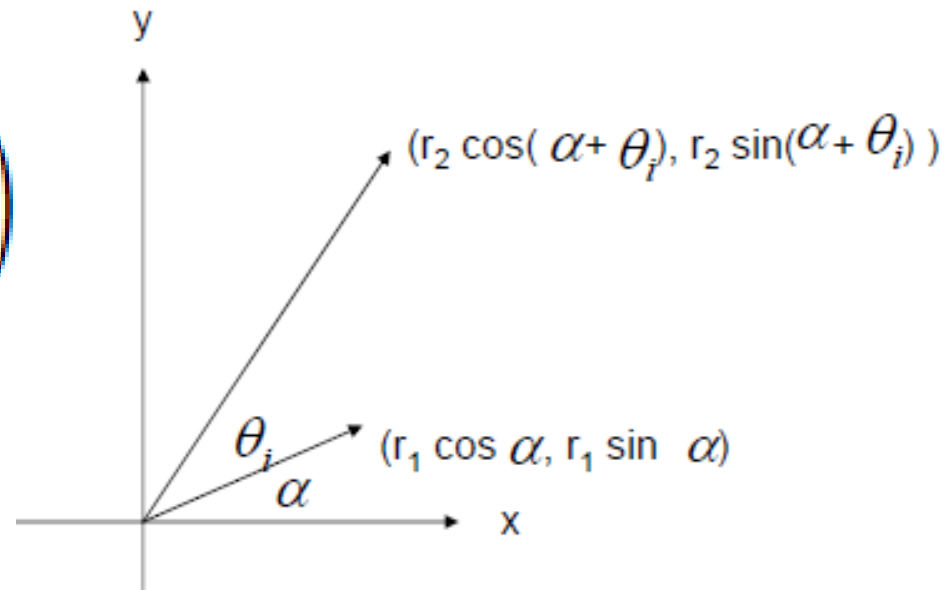
$$f(m, n) = \min_G \max_i \kappa(G_i)$$

- Distributing points on sphere appropriately is usually difficult
 - Stephen Smale @ Berkeley: 18 unsolved problems in 21st century
- The above problem is also difficult for general m and n
 - Give an analytical expression of the optimal code (G) for 2 erasures
 - Explore approximations of G computationally: From practical point view, a numerical approximation of G will work
 - Derive a lower bound for worst case condition number which implicate an upper bound for recovery accuracy in the worst scenario

Optimal codes for 2 erasures

$$G = \begin{pmatrix} g_{11} & \cdots & g_{1n} \\ g_{21} & \cdots & g_{2n} \end{pmatrix}$$

For any sub matrix G_i :



$$G_i = \begin{pmatrix} r_1 \cos \alpha & r_2 \cos(\alpha + \theta_i) \\ r_1 \sin \alpha & r_2 \sin(\alpha + \theta_i) \end{pmatrix}$$

$$G_i^T G_i = \begin{pmatrix} r_1^2 & r_1 r_2 \cos \theta_i \\ r_1 r_2 \cos \theta_i & r_2^2 \end{pmatrix}$$

Optimal codes for 2 erasures

$$G_i^T G_i = \begin{pmatrix} r_1^2 & r_1 r_2 \cos \theta_i \\ r_1 r_2 \cos \theta_i & r_2^2 \end{pmatrix}$$

$$\lambda_{\max}(G_i^T G_i) = \frac{r_1^2 + r_2^2}{2} + \sqrt{\frac{(r_1^2 + r_2^2)^2}{4} + r_1^2 r_2^2 (\cos \theta_i - 1)}$$

$$\lambda_{\min}(G_i^T G_i) = \frac{r_1^2 + r_2^2}{2} - \sqrt{\frac{(r_1^2 + r_2^2)^2}{4} + r_1^2 r_2^2 (\cos \theta_i - 1)}$$

Optimal codes for 2 erasures

$$\begin{aligned}
 \kappa(G_i) &= \sqrt{\frac{\lambda_{\max}(G_i^T G_i)}{\lambda_{\min}(G_i^T G_i)}} \\
 &= \sqrt{\frac{1 + \sqrt{1 + \frac{4(\cos^2 \theta_i - 1)}{\frac{r_1^2}{r_2^2} + \frac{r_2^2}{r_1^2}}}}{1 - \sqrt{1 + \frac{4(\cos^2 \theta_i - 1)}{\frac{r_1^2}{r_2^2} + \frac{r_2^2}{r_1^2}}}}} \\
 &\geq \sqrt{\frac{1 + |\cos \theta_i|}{1 - |\cos \theta_i|}}
 \end{aligned}$$

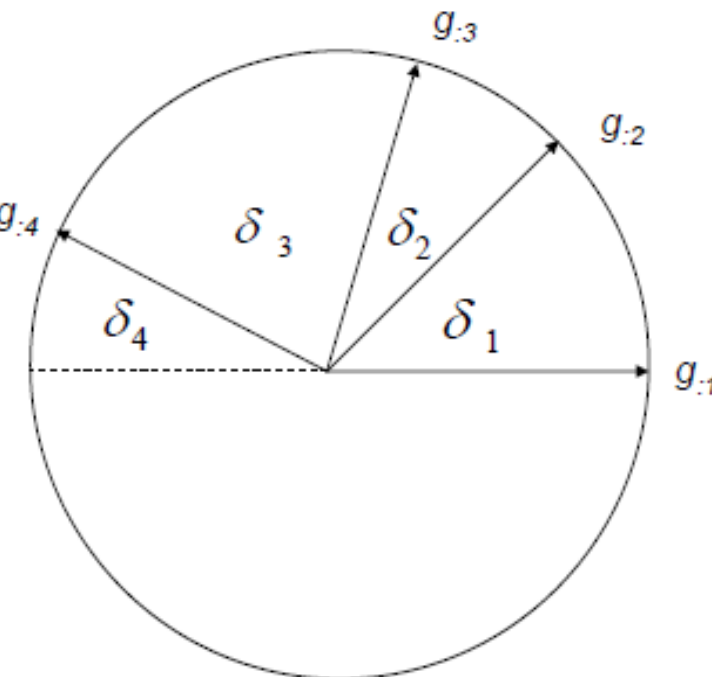
The equality is achieved when $r_1 = r_2$

Optimal codes for 2 erasures

Note that the above relationship exists for any sub-matrix, therefore,

$$\begin{aligned} f(m, n) &= \min_G \max_i \kappa(G_i) \\ &= \min_{\|G(:,j)\|=1} \max_i \kappa(G_i) \end{aligned}$$

Optimal codes for 2 erasures

$$\begin{aligned}
 \min_{\|G(:,j)\|=1} \max_i \kappa(G_i) &= \min_{\delta_1, \dots, \delta_n} \left\{ \max_i \left\{ \sqrt{\frac{1 + |\cos \theta_i|}{1 - |\cos \theta_i|}} \right\} \right\} \\
 &= \min_{\delta_1, \dots, \delta_n} \left\{ \sqrt{\frac{1 + |\cos(\min_i \{\delta_i\})|}{1 - |\cos(\min_i \{\delta_i\})|}} \right\} \\
 &\geq \min_{\delta_1, \dots, \delta_n} \left\{ \sqrt{\frac{1 + |\cos(\frac{\sum_{i=1}^n \delta_i}{n})|}{1 - |\cos(\frac{\sum_{i=1}^n \delta_i}{n})|}} \right\} \\
 &= \sqrt{\frac{1 + \cos(\frac{\pi}{n})}{1 - \cos(\frac{\pi}{n})}}
 \end{aligned}$$


The diagram shows a circle with four vectors, g_1, g_2, g_3, g_4 , originating from the center. The vectors are arranged in a clockwise order starting from the positive x-axis. The angles between consecutive vectors are labeled $\delta_1, \delta_2, \delta_3, \delta_4$. A dashed horizontal line extends from the center to the left edge of the circle, indicating the direction of the negative x-axis.

The equality is achieved when $\delta_1 = \delta_2 = \dots = \delta_n$.

Optimal codes for 2 erasures

$$G = \begin{pmatrix} \cos \frac{\pi}{2n} & \cos \frac{3\pi}{2n} & \cdots & \frac{(2n-1)\pi}{2n} \\ \sin \frac{\pi}{2n} & \cos \frac{3\pi}{2n} & \cdots & \frac{(2n-1)\pi}{2n} \end{pmatrix}$$

$$f(2, n) = \sqrt{\frac{1 + \cos \frac{\pi}{n}}{1 - \cos \frac{\pi}{n}}} \\ \approx \frac{2n}{\pi} \rightarrow \infty \quad (n \rightarrow \infty)$$

- There is **NO** arbitrarily large matrix that satisfies
Any sub-matrix of the matrix is well conditioned
- It is impossible even for the best 2-erasure codes to correct ALL possible 2-erasures when the number of data items (processors) is large
 - The introduced numerical errors can be arbitrarily large

Optimal codes for 2 erasures

$$G = \begin{pmatrix} \cos \frac{\pi}{2n} & \cos \frac{3\pi}{2n} & \cdots & \frac{(2n-1)\pi}{2n} \\ \sin \frac{\pi}{2n} & \cos \frac{3\pi}{2n} & \cdots & \frac{(2n-1)\pi}{2n} \end{pmatrix}$$

$$f(2, n) = \sqrt{\frac{1 + \cos \frac{\pi}{n}}{1 - \cos \frac{\pi}{n}}} \\ \approx \frac{2n}{\pi} \rightarrow \infty \quad (n \rightarrow \infty)$$

- In order to guarantee to correct ALL possible 2-erasures in IEEE standard 754 floating point numbers (16 digits of accuracy) with k digits of accuracy
 - The total number of data items (processors) n must satisfy

$$n \leq 10^{16-k} \cdot \frac{\pi}{2}$$

- When the number of data items (or processors) is larger than $10^{(16-k)}$
 - 100% recovery can be guaranteed by dividing data items (or processors) into subgroup of less than $10^{(16-k)}$ items

Compute Good Codes by Unconstrained Optimization

- Like in Grassmannian frame, in computation practice, we restrict G on matrices whose columns are unit m dimension vectors
 - $\|G(:, j)\| = 1$ for $j=1,2,\dots,n$
- Only m by m sub-matrix will be considered
- Minimax is difficult
 - One possible solution: convert the problem into unconstrained optimization problem
 - But optimality may be lost

Relationship between Condition Number and Determinant

Let $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_m$ denotes the m eigenvalues of $G_j^T G_j$, then

$$\sum_{i=1}^m \lambda_i = \text{tr}(G_j^T G_j) = m \quad \det(G_j^T G_j) = \prod_{i=1}^m \lambda_i$$

$$\begin{aligned} \det G_j &= \sqrt{\det(G_j^T G_j)} \\ &= \sqrt{\prod_{i=1}^m \lambda_i} \\ &= \sqrt{\frac{\lambda_1 \cdot \prod_{i=1}^{m-1} \lambda_i}{\kappa(G_j^T G_j)}} \\ &\leq \sqrt{\frac{\left(\frac{\lambda_1 + \sum_{i=1}^{m-1} \lambda_i}{m}\right)^m}{\kappa(G_j^T G_j)}} \\ &\leq \frac{2^{\frac{m}{2}}}{\kappa(G_j)} \end{aligned}$$

Relationship between Condition Number and Determinant

$$\sum_{i=1}^m \lambda_i = \text{tr}(G_j^T G_j) = m$$

$$\lambda_1 \geq \frac{\sum_{i=1}^m \lambda_i}{m} = 1$$

$$\begin{aligned} \det G_j &= \sqrt{\det(G_j^T G_j)} \\ &= \sqrt{\prod_{i=1}^m \lambda_i} \\ &\geq \sqrt{\lambda_m^m} \\ &\geq \sqrt{\frac{1}{\left(\frac{\lambda_1}{\lambda_m}\right)^m}} \\ &= \frac{1}{\kappa(G_j)^m} \end{aligned}$$

Compute Good Codes by Unconstrained Optimization

$$\frac{1}{\kappa(G_j)^m} \leq \det G_j \leq \frac{2^{\frac{m}{2}}}{\kappa(G_j)}$$

$$\begin{aligned} \det G_j &= \sqrt{\det(G_j^T G_j)} \\ &= \sqrt{\prod_{i=1}^m \lambda_i} \\ &\leq \left(\frac{\sum_{i=1}^m \lambda_i}{m} \right)^m \\ &= 1 \end{aligned}$$

Compute Good Codes by Unconstrained Optimization

$\kappa(G_j)$ is large \iff $\det G_j$ is small

Make all sub-matrices of G well-conditioned \iff make $\prod_i \det G_i$ large

Therefore, instead of solving the original minimax problem, we can solve

$$h(m, n) = \max_{G_{m \times n} \in \mathcal{R}^{m \times n}, \|g_j\|_2 = 1} \left\{ \prod_i \det(G_i) \right\}$$

A Lower Bound for $g(m,n)$

Let ϕ denote the minimum angle between columns of G .

G_j is a sub-matrix containing the two vectors forming the minimum angle.

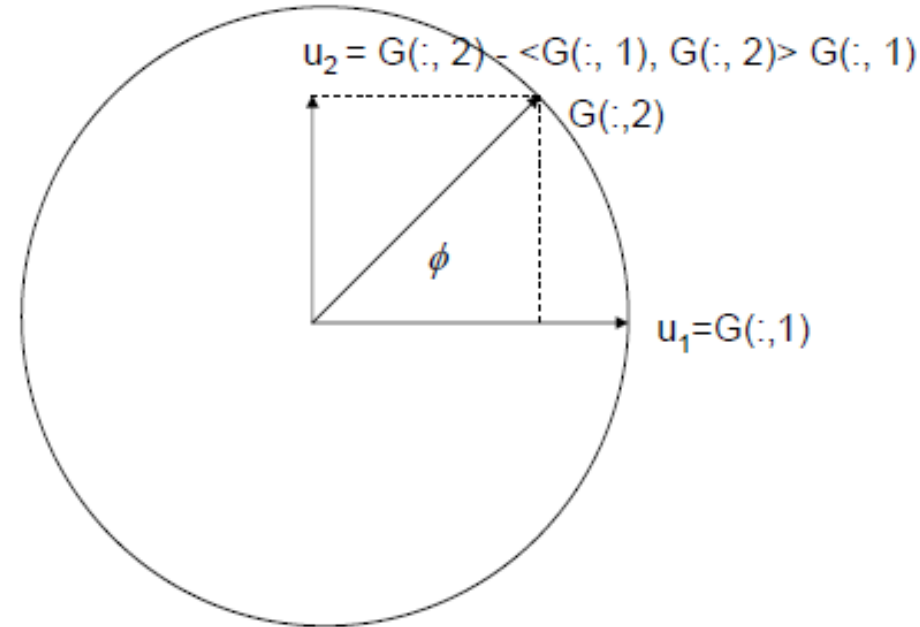
$$\sum_{i=1}^m \lambda_i = \text{tr}(G_j^T G_j) = m \qquad \lambda_1 \geq \frac{\sum_{i=1}^m \lambda_i}{m} = 1$$

$$\begin{aligned} g(m, n) &\geq \kappa(G_j) \\ &= \sqrt{\frac{\lambda_1}{\lambda_m}} \\ &\geq \sqrt{\frac{1}{\lambda_m}} \\ &\geq \sqrt{\frac{1}{(\prod_{i=1}^m \lambda_i)^{\frac{1}{m}}}} \\ &= (\det(G_j^T G_j))^{-\frac{1}{2m}} \\ &= (\det G_j)^{-\frac{1}{m}} \end{aligned}$$

A Lower Bound for $g(m,n)$

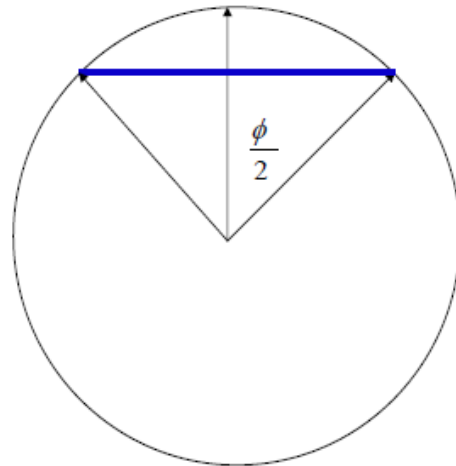
Perform GramSchmidt QR factorization process to orthogonalize $\{G_j(:, 1), G_j(:, 2), \dots, G_j(:, m)\}$ to $\{u_1, u_2, \dots, u_m\}$, then

$$\begin{aligned} \det G_j &= \det(QR) \\ &= \det R \\ &= \prod_{i=1}^m \|u_i\| \\ &\leq \|u_2\| \\ &= \sin \phi \end{aligned}$$



A Lower Bound for $g(m,n)$

Construct n sphere balls $SB^{m-1}(\frac{\phi}{2}, G(:, i))$ on the m -D sphere S^{m-1}
Note that ϕ is the minimum angle between columns of G ,
therefore, there is no overlap between any pair of sphere balls
Let $EB^{m-1}(r, x)$ denote the Euclidean ball in R^m



$$|EB^{m-1}(\sin \frac{\phi}{2}, x)| < |SB^{m-1}(\frac{\phi}{2}, G(:, i))| < \frac{|S^{m-1}|}{n}$$

A Lower Bound for $g(m,n)$

Note that

$$|S^{m-1})| = \frac{2\pi^{m/2}}{\Gamma(m/2)}$$

$$|EB^{m-1}(r, x)| = \frac{\pi^{(m-1)/2} r^{m-1}}{\Gamma((m+1)/2)}$$

Therefore,

$$\sin \frac{\phi}{2} < \left(\frac{2\sqrt{\pi}}{n} \frac{\Gamma(\frac{m}{2} + \frac{1}{2})}{\Gamma(\frac{m}{2})} \right)^{\frac{1}{m-1}}$$

A Lower Bound for $g(m,n)$

Therefore,

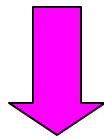
$$\begin{aligned} g(m, n) &\geq (\det G_j)^{-\frac{1}{m}} \\ &\geq (\sin \phi)^{-\frac{1}{m}} \\ &\geq \left(2 \sin \frac{\phi}{2}\right)^{-\frac{1}{m}} \\ &\geq \left(2 \left(\frac{2\sqrt{\pi} \Gamma(\frac{m}{2} + \frac{1}{2})}{n \Gamma(\frac{m}{2})}\right)^{\frac{1}{m-1}}\right)^{-\frac{1}{m}} \\ &= \left(\frac{1}{2}\right)^{\frac{1}{m}} \left(\frac{n \Gamma(\frac{m}{2})}{2\sqrt{\pi} \Gamma(\frac{m}{2} + \frac{1}{2})}\right)^{\frac{1}{m(m-1)}} \\ &\sim n^{\frac{1}{m(m-1)}} \end{aligned}$$

Correcting Errors? Not a Problem!

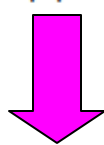
$$A = (I_{n \times n}, G_{m \times n})$$

$$y = Ax$$

$$z = y + e$$



$$\min_{v \in \mathcal{R}^n} \|z - Av\|_{l_1}$$



$$\min_{v \in \mathcal{R}^n, t \in \mathcal{R}^{n+m}} \sum_{i=1}^{n+m} t_i, \quad -t \leq z - Av \leq t$$

Linear Programming:
Polynomial Time

Experimental Results: Worst Case Comparison

Tolerate 3 failures in 10 numbers: 120 possible combinations.

Compare worst 5 condition numbers of all possible 3X3 sub-matrices

Random	47.2541	96.3024	111.9643	161.5512	217.1479
Grassmannian	0.5334*10¹⁷	0.7204*10¹⁷	2.3102* 0¹⁷	2.3102*10¹⁷	Inf
Optimal	14.7503	15.6580	16.1104	16.1609	16.5355

Random:

```
0.2143 -0.2966 0.2635 0.0381 -0.0060 0.0777 -0.2428 -0.0233 0.2121 -0.2288
-0.1691 0.0577 -0.2430 0.1148 0.4060 0.0591 -0.1413 -0.3526 -0.3728 -0.1803
0.3306 0.2845 -0.3741 0.2420 0.4698 -0.4113 0.4380 -0.1750 -0.2025 -0.0038
```

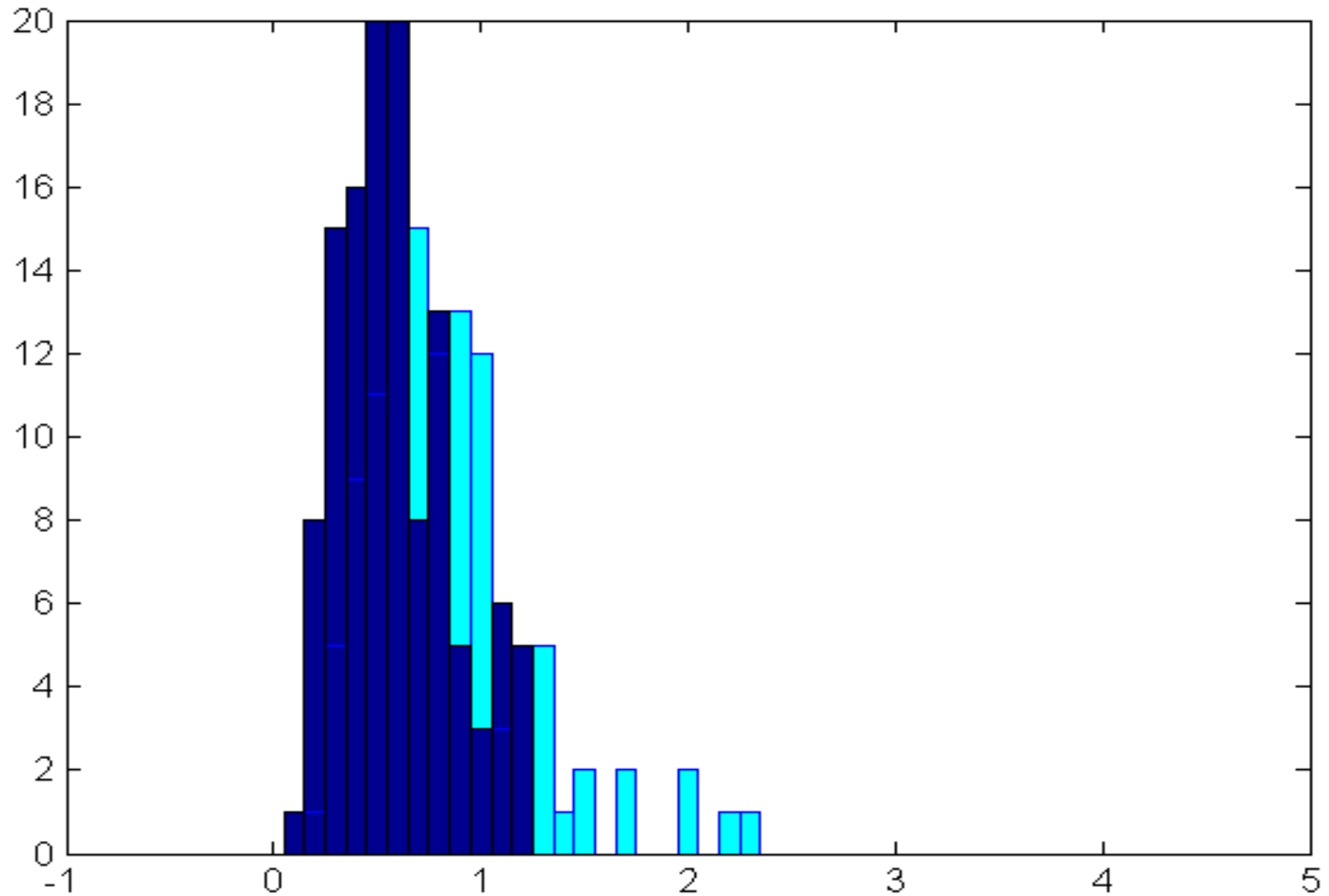
**Grassmannian
(Sloane@A&TT)**

```
1      0.6101 0.6101 0.6101 0.6101 0.6101 0.6101 0      0      0
0      0.7923 0.3961 -0.3961 -0.7923 -0.3961 0.3961 0.8660 -0.8660 0
0      0      0.6861 0.6861 0      -0.6861 -0.6861 0.5000 0.5000 -1
```

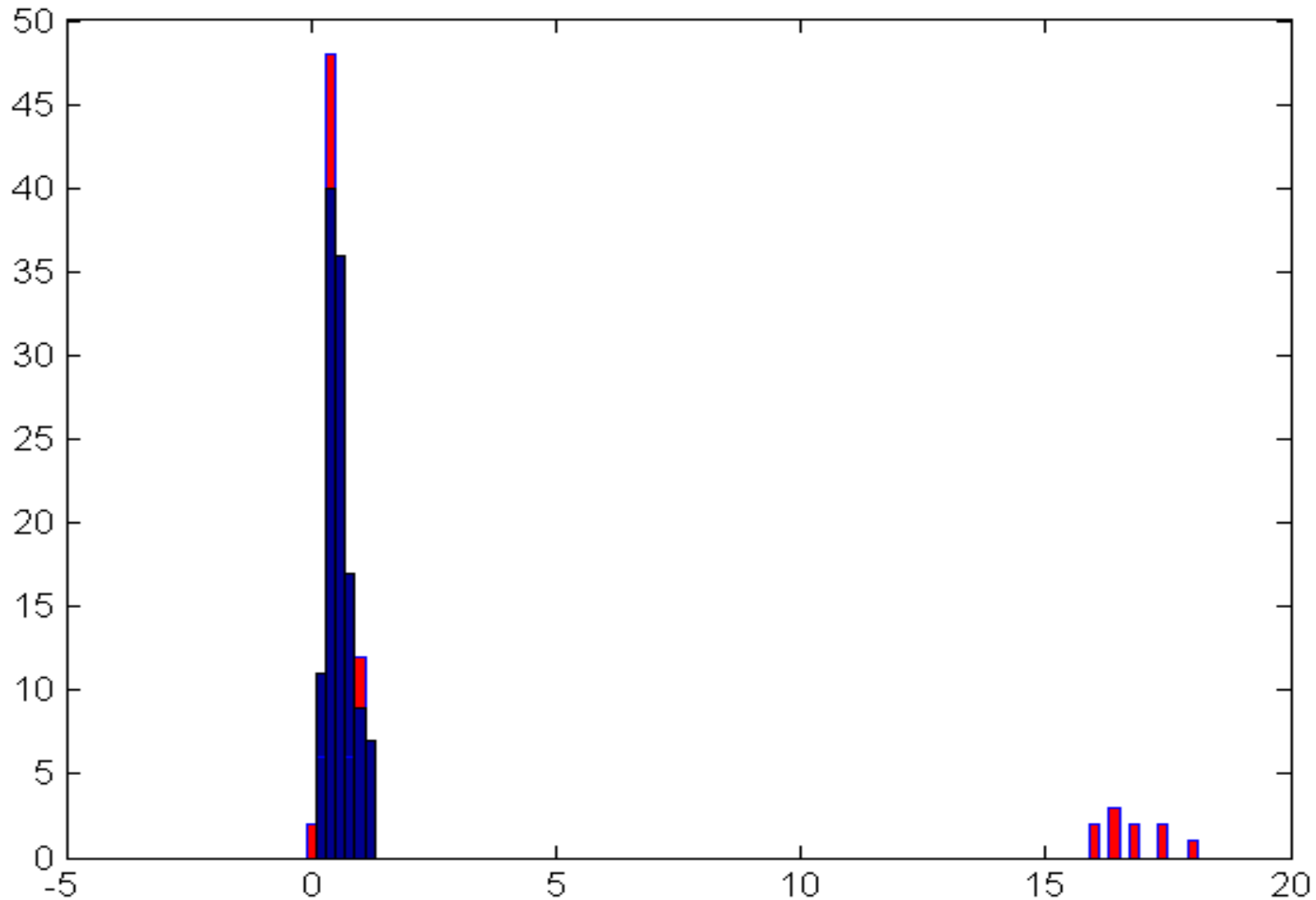
Optimal

```
-0.5566 0.1467 0.7247 0.9919 0.4631 -0.6691 0.5614 -0.2353 0.0686 -0.6749
0.8095 0.7985 0.4905 0.1217 -0.1332 0.2351 -0.6914 -0.0325 0.9466 -0.5804
0.1871 0.5839 0.4839 0.0365 0.8763 0.7050 0.4547 0.9714 -0.3149 0.4556
```

Distribution of all 120 condition numbers: Random vs Optimal



Distribution of all 120 condition numbers: Grassmannian vs Optimal



Questions?