September 29, 2011

High Performance Dense Linear System Solver with Soft Error Resilience

Peng Du, Piotr Luszczek, Jack Dongarra

ICL UT
INNOVATIVE COMPUTING LABORATORY

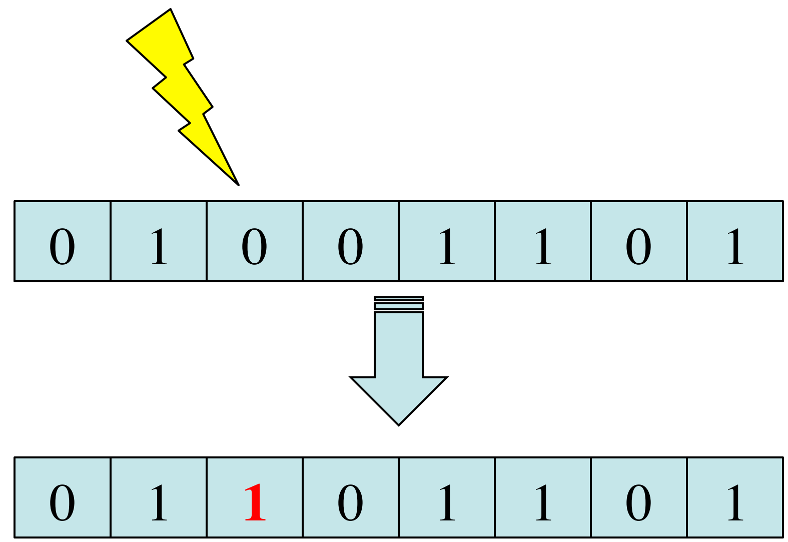
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Agenda

- Soft error threat to the dense linear solver
  - LU factorization
  - Error propagation
- Error modeling
- Fault tolerant algorithm
- Performance Evaluation

Soft error

- Silent error due to radiation
  - Alpha particle
  - High energy neutron
  - Thermal neutron



- Outbreaks
  - Commercial computing system from Sun Microsystem in 2000
  - ASC Q supercomputer at Los Alamos National Lab in 2003

LU based linear solver

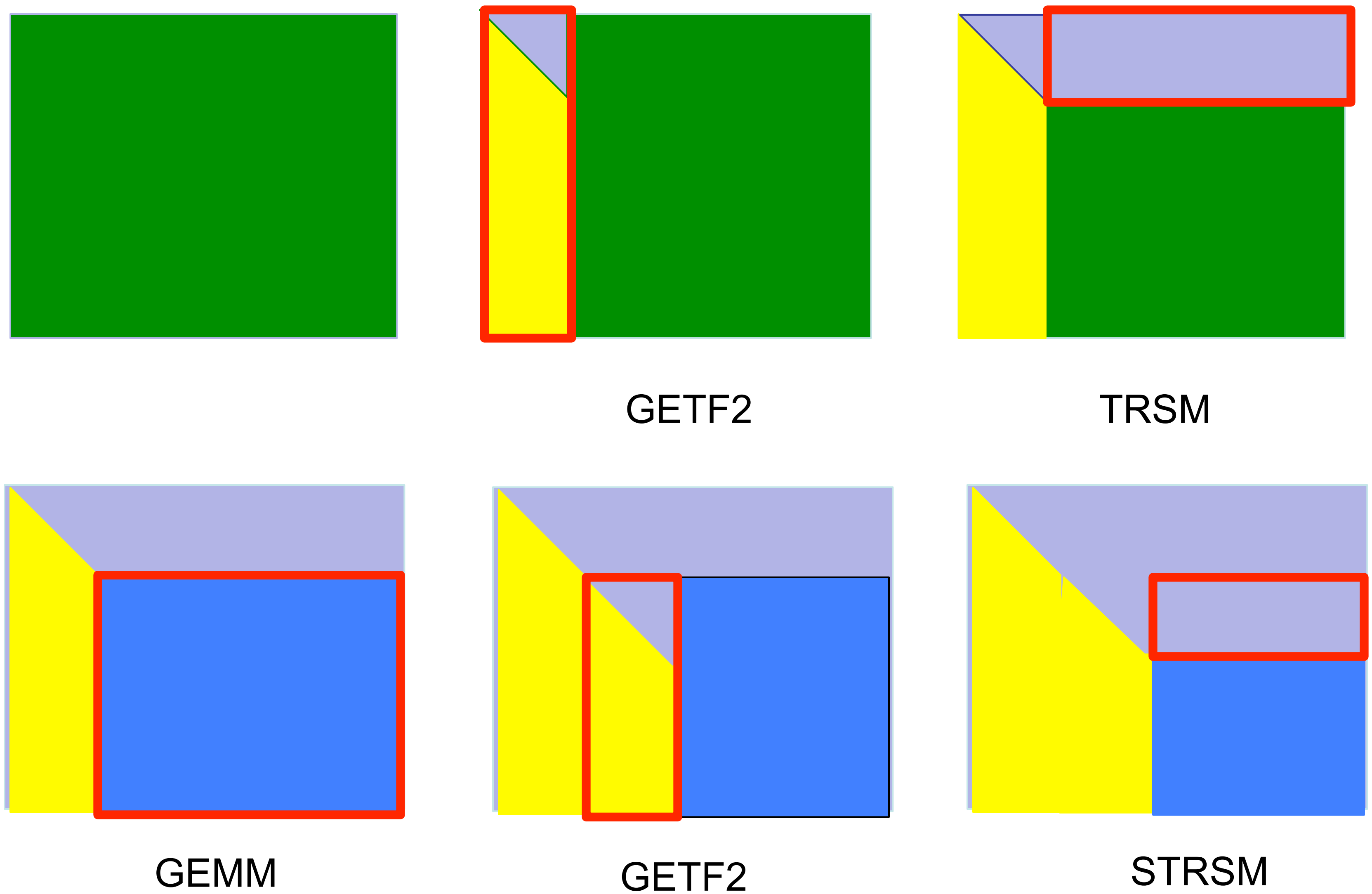
$$Ax = b$$

$$A = LU$$

$$x = U \backslash (L \backslash b)$$

```
>> A=rand(4,4)

A =

    0.6557    0.6787    0.6555    0.2769
    0.0357    0.7577    0.1712    0.0462
    0.8491    0.7431    0.7060    0.0971
    0.9340    0.3922    0.0318    0.8235

>> b=rand(4,1)

b =

    0.6948
    0.3171
    0.9502
    0.0344

>> [L,U]=lu(A)

L =

    0.7021    0.5431    0.9188    1.0000
    0.0382    1.0000         0         0
    0.9091    0.5204    1.0000         0
    1.0000         0         0         0


U =

    0.9340    0.3922    0.0318    0.8235
         0    0.7427    0.1700    0.0147
         0         0    0.5886   -0.6591
         0         0         0    0.2964

>> x=U\(L\b)

x =

    0.4643
    0.3126
    0.5486
   -0.6549

>> norm(A*x-b)

ans =

   1.5701e-16
```

Block LU factorization



General work flow

(1) **Generate checksum** for the input matrix as additional columns

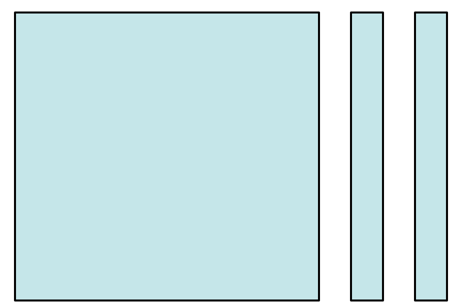
(2) Perform **LU factorization** WITH the additional checksum columns

(3) **Solve *Ax=b*** using LU from the factorization

(even if soft error occurs during LU factorization)

(4) **Check** for soft error

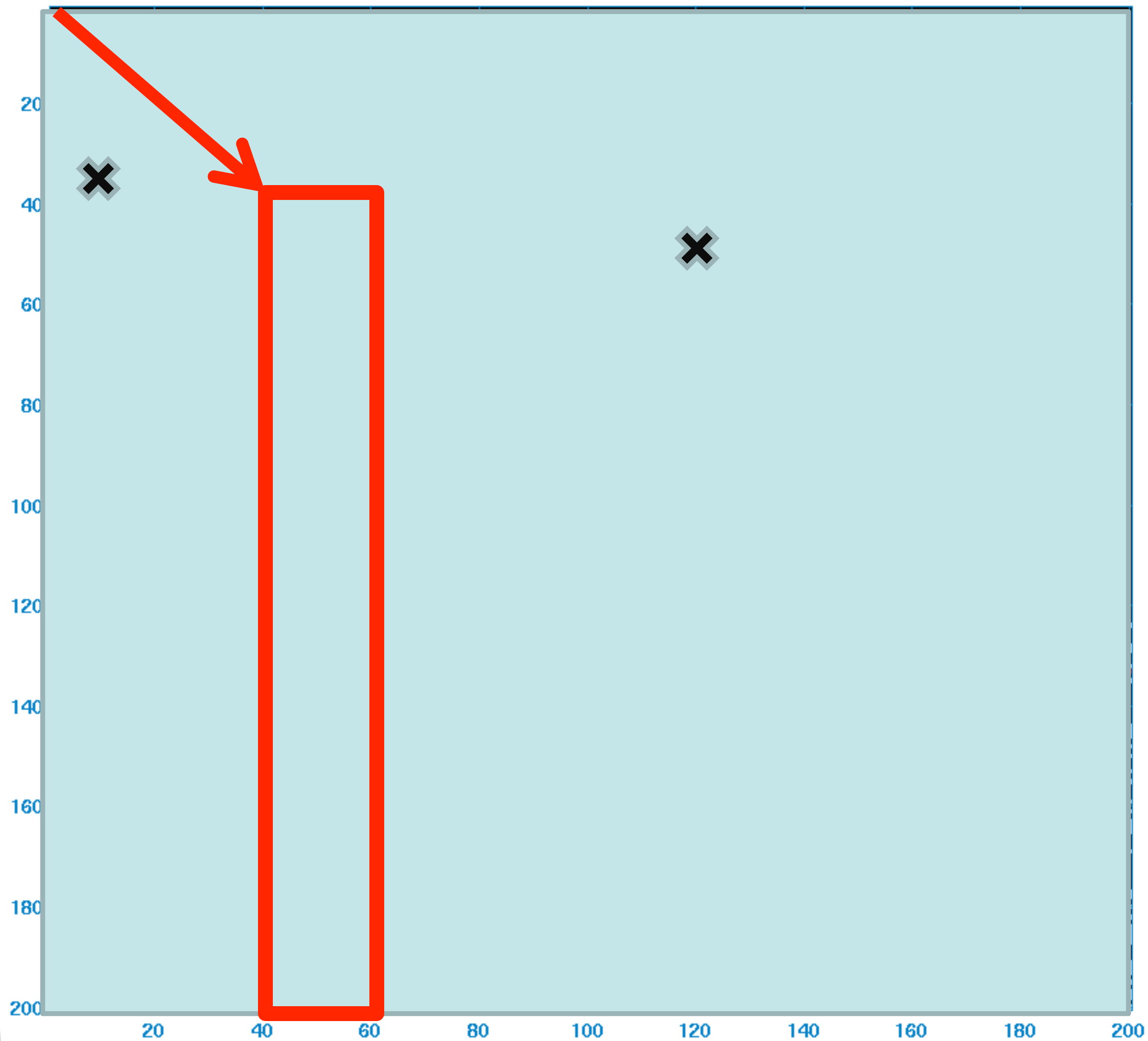
(5) **Correct** solution *x*



Why is soft error hard to handle?

- Soft error occurs silently
- Propagation

Example: Error propagation



Error location (using matlab notation and 1-based index)

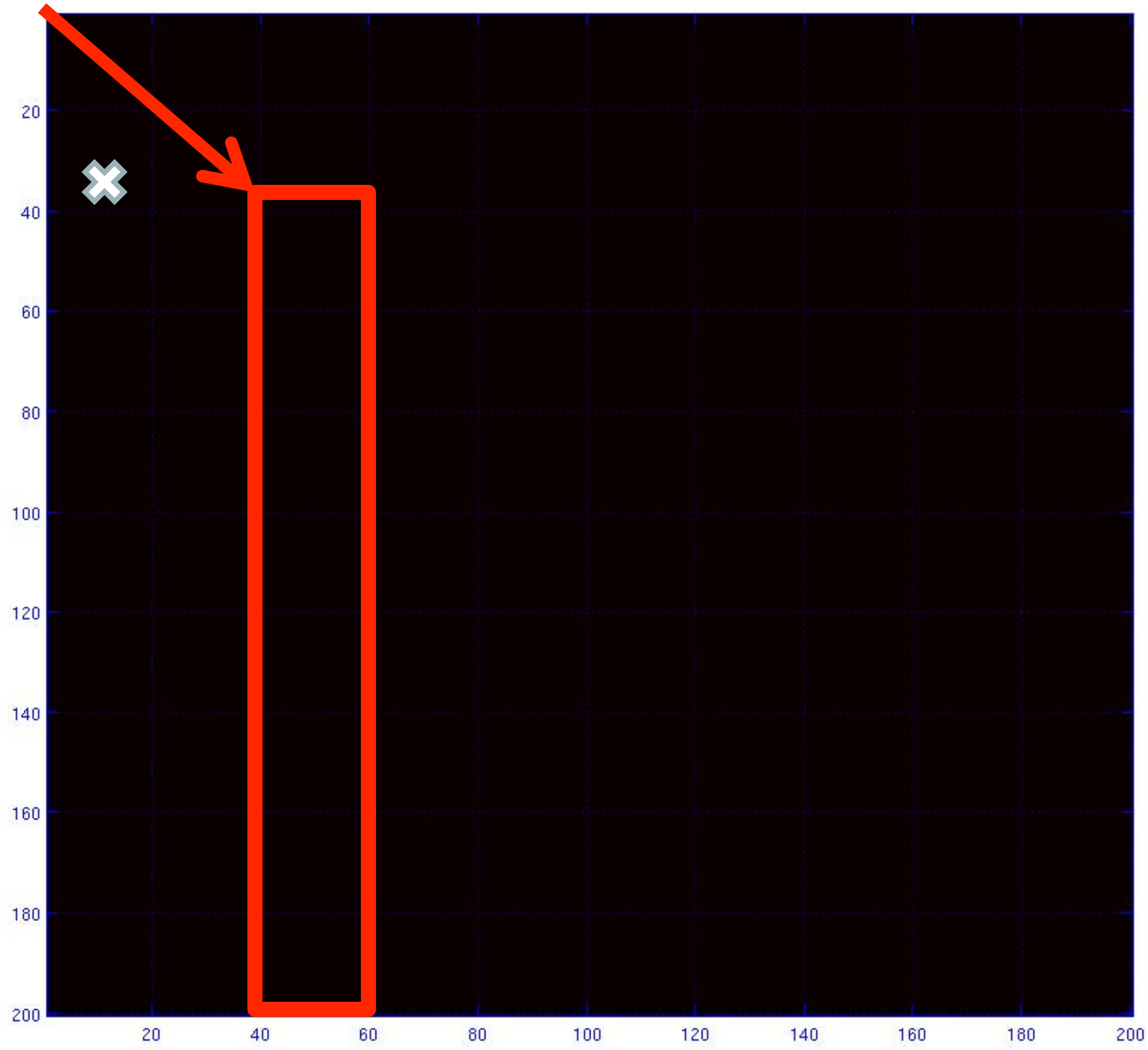
Error strikes right before panel factorization of (41:200, 41:60),

Case 1: Error at (35,10), in L area

Case 2: Error at (50,120), in A' area

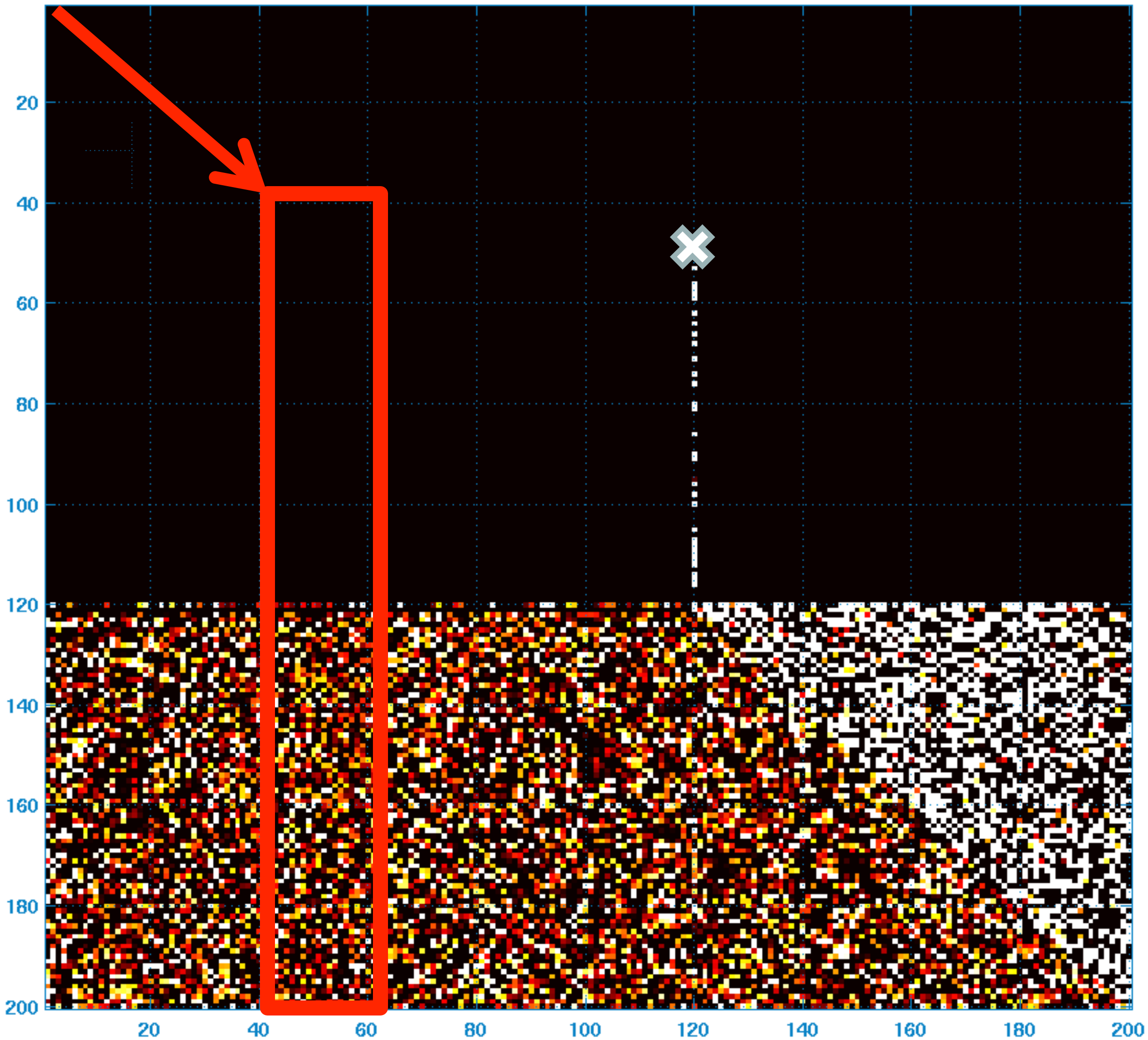
Note: Pivoting on the left of panel factorization is delayed to the end of error detection and recovery so that error in L area does not move

Case 1: Non-propagating error



Error does not propagate in this case

Case 2: Propagating error



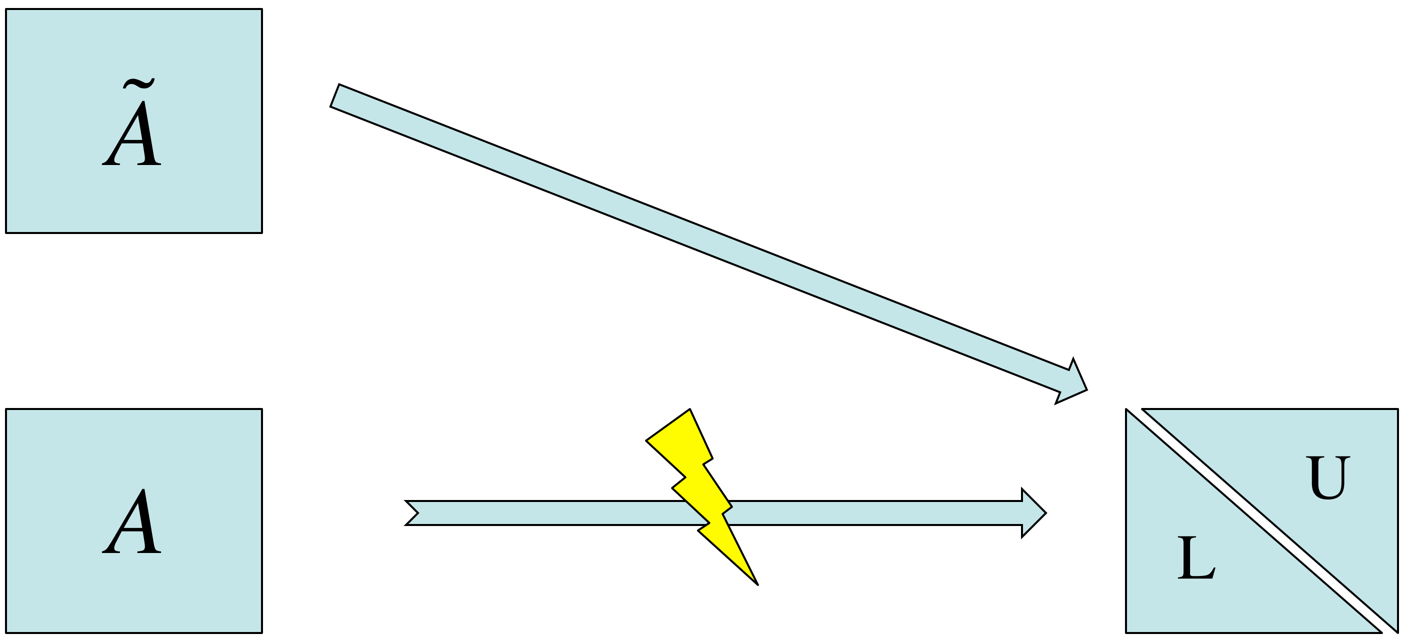
Soft error challenge

When?

Where?

Error modeling
(for propagating error)

- When?
  - Answer: ***Doesn't really matter***



Error modeling (for "where")

Input matrix A

One step of LU $A_t = L_{t-1}P_{t-1}A_{t-1}$

If no soft error occurs $U = (L_n P_n)\cdots(L_1 P_1)(L_0 P)_0 A_0$

If soft error occurs at step t $\tilde{A}_t = L_{t-1}P_{t-1}A_{t-1} - \lambda e_i e_j^T$

$$= L_{t-1}P_{t-1}(L_{t-2}P_{t-2}\cdots L_0 P_0)A_0 - \lambda e_i e_j^T$$

Define an initial erroneous initial matrix $\tilde{A}$

$$\tilde{A} \cong (L_{t-1}P_{t-1}L_{t-2}P_{t-2}\cdots L_0 P_0)^{-1}\tilde{A}_t$$

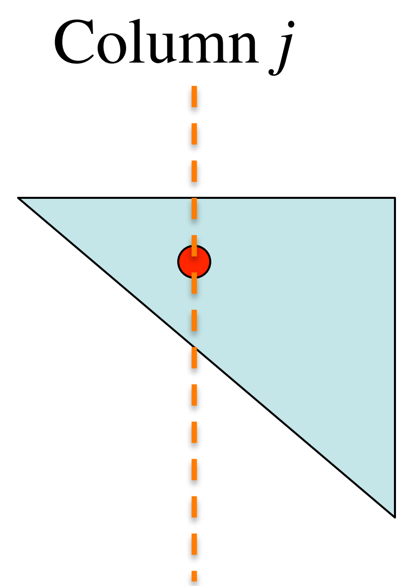
$$= A - (L_{t-1}P_{t-1}L_{t-2}P_{t-2}\cdots L_0 P_0)^{-1}\lambda e_i e_j^T = A - de_j^T$$

Locate Error

$$\tilde{P}[\tilde{A}, A \times e, A \times w] = \tilde{L}[\tilde{U}, \tilde{c}, \tilde{v}], \quad \tilde{A} = A + de_j^T$$

$$\Rightarrow \tilde{P}[\tilde{A}, Ae, Aw] = \tilde{L}[\tilde{U}, \tilde{c}, \tilde{v}]$$

$$\Rightarrow \begin{cases} \tilde{P}\tilde{A} = \tilde{L}\tilde{U} \\ \tilde{P}Ae = \tilde{L}\tilde{c} \\ \tilde{P}Aw = \tilde{L}\tilde{v} \end{cases}$$



$$G = \begin{bmatrix} e^T \\ w^T \end{bmatrix} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ w_1 & w_2 & \cdots & w_n \end{bmatrix}^T$$

Recover Ax=b

- Luk's work
- Sherman Morison Formula

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^T A^{-1}}{1 + v^T A^{-1}u}.$$

Recover Ax=b

Given:

$$\begin{cases} \tilde{P}\tilde{A} = \tilde{L}\tilde{U} \\ \tilde{A}\tilde{x} = b \end{cases}$$

To Solve:

$$Ax = b$$

Recover Ax=b

$$Ax = b$$

$$\Rightarrow \quad x = A^{-1}b$$

$$\Rightarrow \quad x = A^{-1}(\tilde{P}^{-1}\tilde{P})b = (\tilde{P}A)^{-1}\tilde{P}b$$

$$(\tilde{P}A)^{-1} = ?$$

Recover Ax=b

Recall:

$$A - \tilde{A} = de_j^T$$

Therefore:

$$\tilde{P}A - \tilde{P}\tilde{A} = (\tilde{P}a_{\bullet j} - \tilde{L}\tilde{U}_{\bullet j})e_j^T$$

$$\tilde{P}A = \tilde{L}\tilde{U} + \tilde{L}(\tilde{L}^{-1}\tilde{P}a_{\bullet j} - \tilde{U}_{\bullet j})e_j^T = \tilde{L}(\tilde{U} + te_j^T)$$

$$= \tilde{L}\tilde{U}(I + \tilde{U}^{-1}te_j^T) = \tilde{L}\tilde{U}(I + ve_j^T)$$

$$t = \tilde{L}^{-1}\tilde{P}a_{\bullet j} - \tilde{U}_{\bullet j}$$

$$v = \tilde{U}^{-1}t$$

Recover Ax=b

$$(\tilde{P}A)^{-1} = (\tilde{L}\tilde{U}(I + ve_j^T))$$

$$= (I + ve_j^T)^{-1}(\tilde{L}\tilde{U})^{-1}$$

Sherman Morrison

$$= \left(I - \frac{1}{1 + v_j} ve_j^T \right)(\tilde{L}\tilde{U})^{-1}$$

Recover Ax=b

$$Ax = b$$

$$= \left(I - \frac{1}{1 + v_j} v e_j^T \right) \tilde{x}$$

Recover Ax=b

$$(1) \quad \tilde{L}\tilde{U}\tilde{x} = \tilde{P}b$$

$$(2) \quad \begin{cases} t = \tilde{L}^{-1}\tilde{P}a_{\bullet j} - \tilde{U}_{\bullet j} \\ v = \tilde{U}^{-1}t \\ x = \left(I - \dfrac{y_j}{1+v_j} v e_j^T \right) \tilde{x} \end{cases}$$

Recover Ax=b

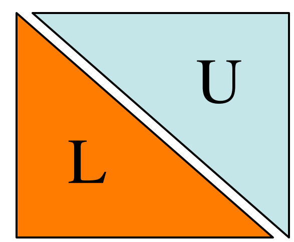
$$(1) \quad \tilde{L}\tilde{U}\tilde{x} = \tilde{P}b$$

Needs protection

$$(2) \quad \begin{cases} t = \tilde{L}^{-1}\tilde{P}a_{\bullet j} - \tilde{U}_{\bullet j} \\ v = \tilde{U}^{-1}t \\ x = \left(I - \dfrac{y_j}{1+v_j} v e_j^T\right)\tilde{x} \end{cases}$$

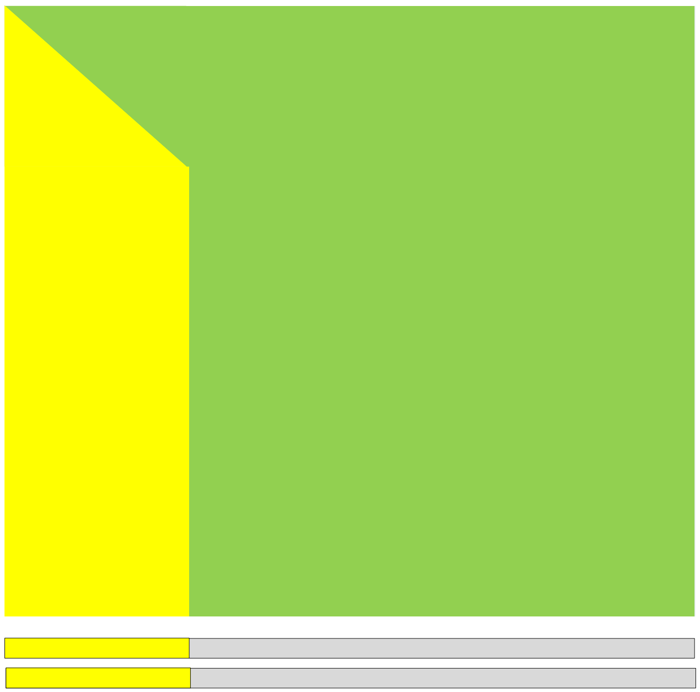
How to detect & recovery a soft error in L?

- The recovery of Ax=b requires a correct L
- L does not change once produced
  - Static checkpointing for L
- Delay pivoting on L to prevent checksum of L from being invalidated



Checkpointing for L, idea 1

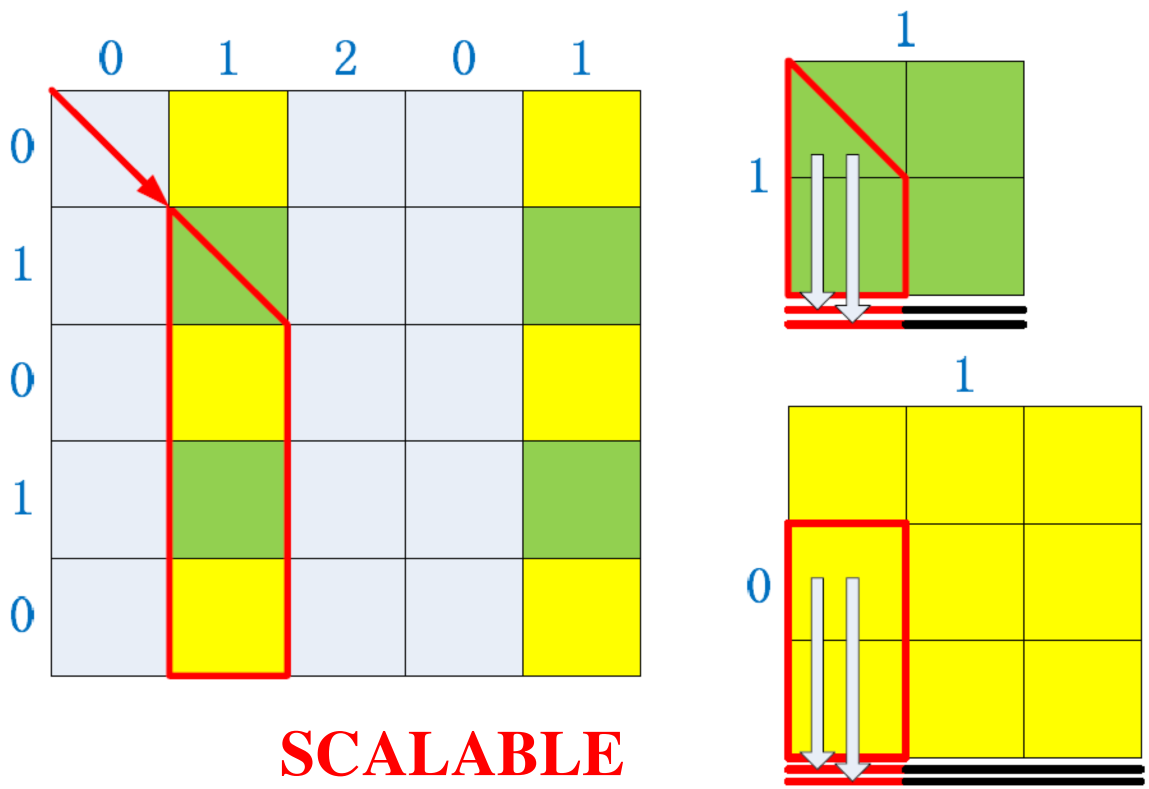
- PDGEMM based checkpointing
- Checkpointing time increases when scaled to more processes and larger matrices



NOT SCALABLE

Checkpointing for L, idea 2

- Local Checkpointing
- Each process checkpoints their local involved data
- Constant checkpointing time



Encoding for L

- On each process, for a column of L $l = [l_1, l_2, \cdots, l_n]$

$$\begin{cases} l_1 + l_2 + \cdots + l_n = c_1 \\ w_1 l_1 + w_2 l_2 + \cdots + w_n l_n = c_2 \end{cases}$$

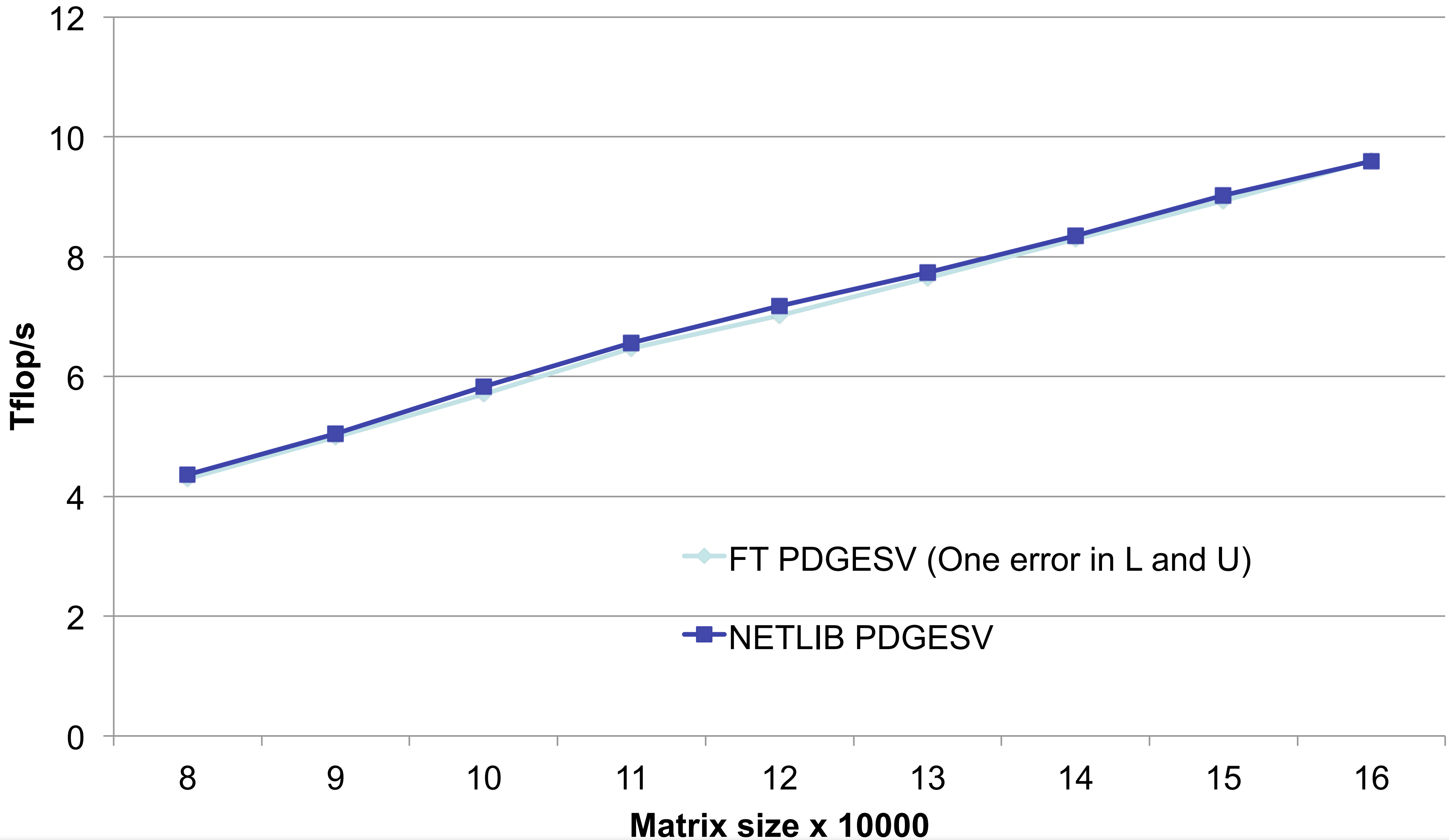
$$\begin{cases} l_1 + \cdots + \tilde{l}_i + \cdots + l_n = \tilde{c}_1 \\ w_1 l_1 + \cdots + w_i \tilde{l}_i + \cdots + w_n l_n = \tilde{c}_2 \end{cases}$$

$$\begin{cases} c_1 - \tilde{c}_1 = l_i - \tilde{l}_i \\ c_2 - \tilde{c}_2 = w_i (l_i - \tilde{l}_i) \end{cases} \Rightarrow \quad w_i = \frac{c_2 - \tilde{c}_2}{c_1 - \tilde{c}_1}$$

Kraken Performance

Two 2.6 GHz six-core AMD Opteron processors per node

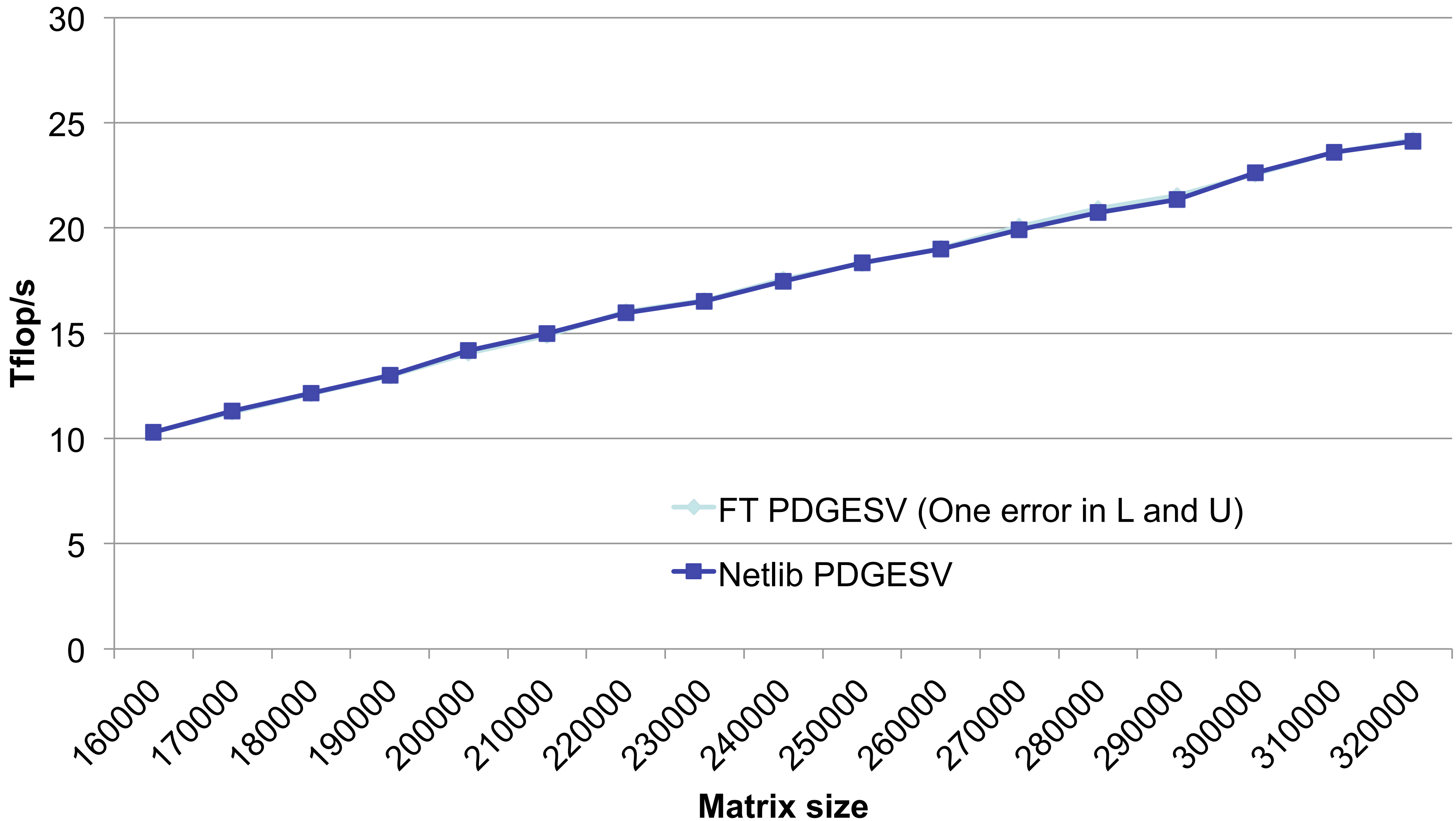
32x32 MPI processes, **6** threads/(process, core)
6,144 cores used in total



Kraken Performance

Two 2.6 GHz six-core AMD Opteron processors per node

64x64 MPI processes, **6** threads/(process, core)
24,576 used cores in total



Quəstion?

- Backup slides

Locate Error

$$\tilde{P}Ae = \tilde{L}\tilde{c}$$

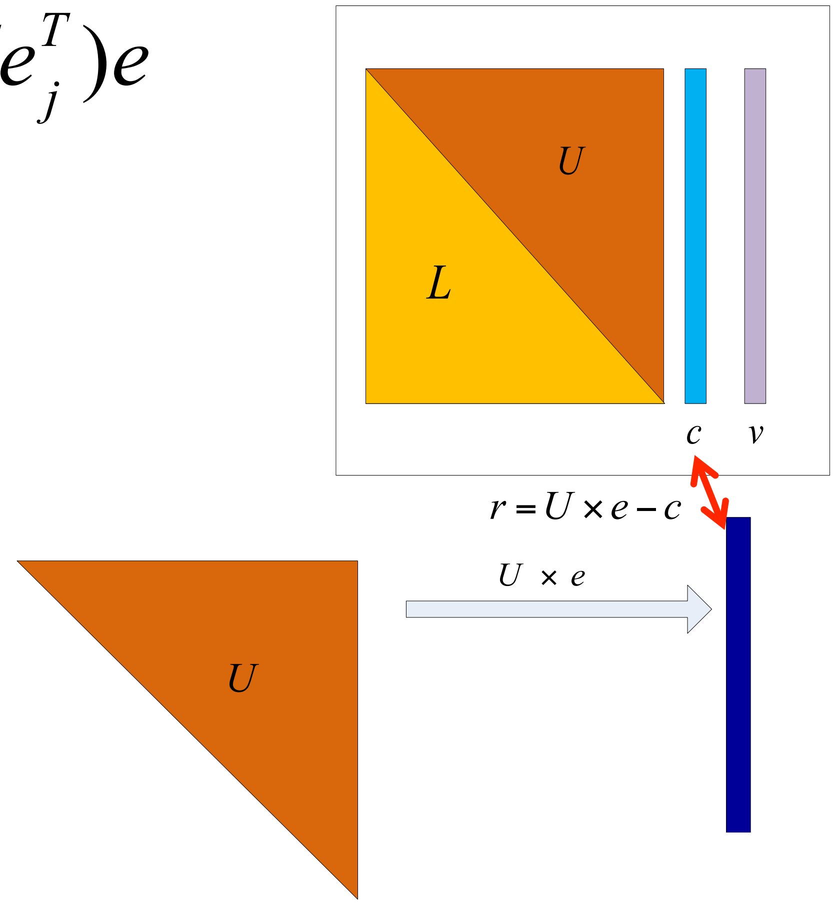
$$\Rightarrow \quad \tilde{c} = \tilde{L}^{-1}\tilde{P}Ae = \tilde{L}^{-1}\tilde{P}(\tilde{A} + de_j^T)e$$

$$= \tilde{L}^{-1}(\tilde{P}\tilde{A} + \tilde{P}de_j^T)e$$

$$= \tilde{L}^{-1}(\tilde{L}\tilde{U} + \tilde{P}de_j^T)e$$

$$= \tilde{U}e + \tilde{L}^{-1}\tilde{P}d$$

$$\Rightarrow \quad \tilde{c} - \tilde{U}e = \tilde{L}^{-1}\tilde{P}d = r$$



Locate Error

$$\tilde{P}Aw = \tilde{L}\tilde{v}$$

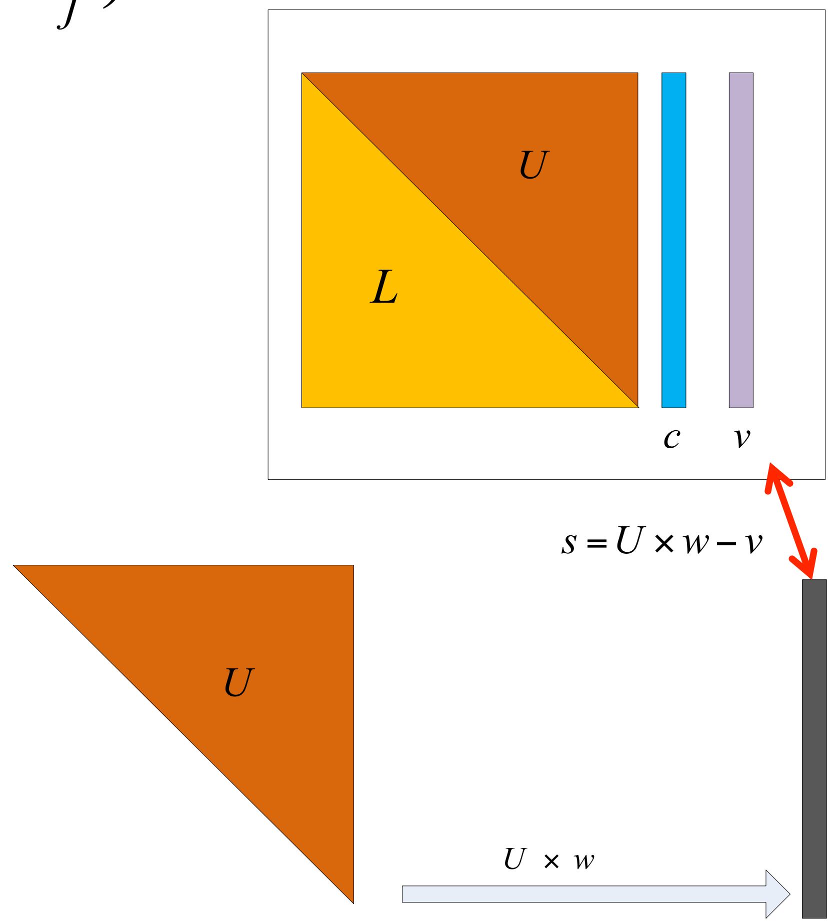
$$\Rightarrow \quad \tilde{v} = \tilde{L}^{-1}\tilde{P}Aw = \tilde{L}^{-1}\tilde{P}(\tilde{A} + de_j^T)w$$

$$= \tilde{L}^{-1}(\tilde{P}\tilde{A} + \tilde{P}de_j^T)w$$

$$= \tilde{L}^{-1}(\tilde{L}\tilde{U} + \tilde{P}de_j^T)w$$

$$= \tilde{U}w + \tilde{L}^{-1}\tilde{P}dw_j$$

$$\Rightarrow \quad \tilde{v} - \tilde{U}w = \tilde{L}^{-1}\tilde{P}dw_j = s$$



Locate Error

$$\begin{cases} \tilde{c} - \tilde{U}e = \tilde{L}^{-1}\tilde{P}d = r \\ \tilde{v} - \tilde{U}w = w_j \tilde{L}^{-1}\tilde{P}d = s \end{cases} \Rightarrow \quad s = w_j \times r$$

$$\Rightarrow w_j \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = s./r$$

- W_j is the j_{th} element of vector w in the generator matrix
- Component-wise division of s and r reveals w_j
- Search w_j in w reveals the **initial soft error's column**

Extra Storage

- For input matrix of size MxN on PxQ grid
  - A copy of the original matrix
    - Not necessary when it's easy to re-generate the required column of the original matrix
  - 2 additional columns: 2 x M
  - Each process has 2 rows: $2 \times \frac{N}{Q}$, in total $P \times 2 \times N$

$$r = \frac{extra\ storage}{matrix\ storage} = \frac{2 \times M + P \times 2 \times N}{M \times N}$$

$$= \frac{2}{N} + \frac{P \times 2}{M} \xrightarrow{N \to \infty} \frac{P \times 2}{M}$$