Convergence and Soft Fault Resilience for Fine-Grained Parallel Incomplete Factorizations

Non-symmetric Problems

Evan Coleman and Masha Sosonkina

High Performance Computing Symposium
SpringSim '18

Acknowledgements

This work was supported in part by:

- Air Force Office of Scientific Research under the AFOSR award FA9550-12-1-0476
- U.S. Department of Energy (DOE) Office of Advanced Scientific Computing Research under the grant DE-SC-0016564 and through the Ames Laboratory, operated by Iowa State University under contract No. DE-AC00-07CH11358
- U.S. Department of Defense High Performance Computing Modernization Program, through a HASI grant
- ILIR/IAR program at NSWC Dahlgren
- Turing High Performance Computing cluster at Old Dominion University

Outline

Outline

Fine-Grained Parallel Incomplete LU (FGPILU) Factorization

- Given a sparse matrix, A, compute factors L and U such that,

$$A \approx LU \tag{1}$$

Fine-Grained Parallel Incomplete LU (FGPILU) Factorization

- Given a sparse matrix, A, compute factors L and U such that,

$$A \approx LU \tag{1}$$

- Define the sparsity pattern as,

$$S = \{(i,j)|l_{ij} \neq 0 or u_{ij} \neq 0\} \tag{2}$$

Fine-Grained Parallel Incomplete LU (FGPILU) Factorization

- Given a sparse matrix, A, compute factors L and U such that,

$$A \approx LU \tag{1}$$

- Define the sparsity pattern as,

$$S = \{(i,j) | l_{ij} \neq 0 \text{or} u_{ij} \neq 0\} \tag{2}$$

- Chow and Patel[1] make the observation that,

$$(LU)_{ij} = a_{ij} \tag{3}$$

for $(i,j) \in S$

[1] Edmond Chow and Aftab Patel. "Fine-grained parallel incomplete LU factorization". In: *SIAM journal on Scientific Computing* 37.2 (2015), pp. C169–C193.

Fine-Grained Parallel Incomplete LU (FGPILU) Factorization

- This allows for the components of the L and U factors to be solved for iteratively, in place of using a traditional Gaussian elimination style approach.

Fine-Grained Parallel Incomplete LU (FGPILU) Factorization

- This allows for the components of the L and U factors to be solved for iteratively, in place of using a traditional Gaussian elimination style approach.
- To do this, we make use of the constraint,

$$\sum_{k=1}^{\min(i,j)} l_{ik} u_{kj} = a_{ij} \tag{4}$$

for $(i, j) \in S$. This gives $|S|$ unknowns and $|S|$ constraints.

Fine-Grained Parallel Incomplete LU (FGPILU) Factorization

- This leads to two nonlinear equations:
  1. $l_{ij} = \frac{1}{u_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \right)$,
  2. $u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}$.

Fine-Grained Parallel Incomplete LU (FGPILU) Factorization

- This leads to two nonlinear equations:
  1. $l_{ij} = \frac{1}{u_{jj}}\left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}\right)$,
  2. $u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}$.
- These equations can be used to find the l_{ij} and u_{ij} components of L and U via a fixed point iteration,

$$x^{k+1} = G(x^k) \tag{5}$$

  where G captures the two equations above and an initial guess x^0 is supplied.
- This allows a higher degree of parallelism where all of the components can be updated in parallel.

Convergence of the FGPILU Factorization

- Convergence of nonlinear fixed point iterations is related to both the spectral radius and norm of the associated Jacobian matrix.
- In order to define the Jacobian, an ordering of the elements in both L and U needs to be defined. Let $g(i,j)$ be an ordering that takes every index in L and U and maps it to $1, 2, 3, \ldots, (nnz(L) + nnz(U))$
- This allows the two nonlinear equations to be rewritten such that the fixed point iteration becomes,

$$G_{g(i,j)} = \begin{cases} \frac{1}{x_{g(j,j)}} \left(a_{ij} - \sum\limits_{1 \le k \le j-1} x_{g(i,k)} x_{g(k,j)} \right) & i > j \\ a_{ij} - \sum\limits_{1 \le k \le i-1} x_{g(i,k)} x_{g(k,j)} & i \le j \end{cases} \tag{6}$$

FGPILU Jacobian

- The Jacobian itself is defined by the following equations:

$$\frac{\partial G_{g(i,j)}}{\partial x_{g(k,j)}} = -\frac{x_{g(i,k)}}{x_{g(j,j)}}, k < j \qquad \frac{\partial G_{g(i,j)}}{\partial x_{g(i,k)}} = -x_{g(i,k)}, k < i$$

$$\frac{\partial G_{g(i,j)}}{\partial x_{g(i,k)}} = -\frac{x_{g(k,j)}}{x_{g(j,j)}}, k < j \qquad \frac{\partial G_{g(i,j)}}{\partial x_{g(k,j)}} = -x_{g(i,k)}, k < i$$

$$\frac{\partial G_{g(i,j)}}{\partial x_{g(j,j)}} = -\frac{1}{x_{g(j,j)}^2}\left(a_{ij} - \sum_{k=1}^{j-1} x_{g(i,k)}x_{g(k,j)}\right)$$

- Equations in the left column are for a row in the Jacobian where $i > j$ ($l_{ij} \in L$), and equations in the right column for a row $i \leq j$ ($u_{ij} \in U$).

Fine-grained Methods

- Overview of fine-grained methods:
  - Can operate in synchronous environments or asynchronous environments
  - May be better suited for computation on accelerators (i.e. GPUs)
  - Allows for component level checking on accuracy of solution and existence of faults

Fine-grained Methods

- Overview of fine-grained methods:
  - Can operate in synchronous environments or asynchronous environments
  - May be better suited for computation on accelerators (i.e. GPUs)
  - Allows for component level checking on accuracy of solution and existence of faults
- Outline of goals for fine-grained methods:
  - Each component (or block of components) can be treated as a task
  - It is able to be assigned to any given processor
  - Each processor should be able to complete its current task without receiving new information from other processors
  - Information (possibly stale) may be required concerning the state of other components

Fine-Grained Fixed-Point Iteration

- The fixed point iteration $x = G(x)$ can be broken into the individual component functionals $g_i(x)$ that update each element of x where $x = G(x) = (g_1(x), g_2(x), \cdots, g_n(x))$:

$$
\begin{aligned}
x_1 &= g_1(x_1, x_2, x_3, \cdots, x_n) \\
x_2 &= g_2(x_1, x_2, x_3, \cdots, x_n) \\
x_3 &= g_3(x_1, x_2, x_3, \cdots, x_n) \\
&\vdots \\
x_n &= g_n(x_1, x_2, x_3, \cdots, x_n)
\end{aligned}
$$

Fine-Grained Fixed-Point Iteration

- The fixed point iteration $x = G(x)$ can be broken into the individual component functionals $g_i(x)$ that update each element of x where $x = G(x) = (g_1(x), g_2(x), \cdots, g_n(x))$:

$$
\begin{aligned}
x_1 &= g_1(x_1, x_2, x_3, \cdots, x_n) \\
x_2 &= g_2(x_1, x_2, x_3, \cdots, x_n) \\
x_3 &= g_3(x_1, x_2, x_3, \cdots, x_n) \\
&\vdots \\
x_n &= g_n(x_1, x_2, x_3, \cdots, x_n)
\end{aligned}
$$

- In the synchronous case, all updates $g_i(x)$ use the same values for $x_i \in x$, whereas in the asynchronous case each call to a g_i uses the latest values of x_i that are available.
    - Note: each g_i may be responsible for updating a single element, or a block of elements.

Outline

FGPILU Convergence

- Local convergence is guaranteed in both the synchronous and asynchronous case if the spectral radius, ρ, satisfies $\rho(|G'(x^*)|) < 1$ where x^* is the fixed point of G and G' represents the Jacobian of G

FGPILU Convergence

- Local convergence is guaranteed in both the synchronous and asynchronous case if the spectral radius, ρ, satisfies $\rho(|G'(x^*)|) < 1$ where x^* is the fixed point of G and G' represents the Jacobian of G
- Given a Gaussian elimination style ordering, $G'(x)$ has zeros on the diagonal and therefore a spectral radius of 0 for any x
    - This gives local convergence trivially
    - Global convergence results exist[2], but are not practically helpful

[2] Chow and Patel, "Fine-grained parallel incomplete LU factorization".

Convergence of the FGPILU Algorithm

- The goal is to increase both the ability of the FGPILU fixed point iteration to converge, and to increase the rate at which it does.
- The partial derivatives in the Jacobian suggest that increasing the diagonal dominance of the matrix will improve the convergence of the FGPILU algorithm.

Convergence of the FGPILU Algorithm

- The goal is to increase both the ability of the FGPILU fixed point iteration to converge, and to increase the rate at which it does.
- The partial derivatives in the Jacobian suggest that increasing the diagonal dominance of the matrix will improve the convergence of the FGPILU algorithm.
- Two main tracks of ideas:

Convergence of the FGPILU Algorithm

- The goal is to increase both the ability of the FGPILU fixed point iteration to converge, and to increase the rate at which it does.
- The partial derivatives in the Jacobian suggest that increasing the diagonal dominance of the matrix will improve the convergence of the FGPILU algorithm.
- Two main tracks of ideas:
  - Reordering the original input matrix, recovering the solution after preconditioning, and solving the preconditioned linear system.

Convergence of the FGPILU Algorithm

- The goal is to increase both the ability of the FGPILU fixed point iteration to converge, and to increase the rate at which it does.
- The partial derivatives in the Jacobian suggest that increasing the diagonal dominance of the matrix will improve the convergence of the FGPILU algorithm.
- Two main tracks of ideas:
    - Reordering the original input matrix, recovering the solution after preconditioning, and solving the preconditioned linear system.
    - Applying the preconditioning algorithm to a modified matrix with artificially increased diagonal dominance and using the resultant preconditioner on the original matrix.

Matrix Reorderings

- Four reorderings were used:

Matrix Reorderings

- Four reorderings were used:
  1. Natural ordering
     - No changes to the ordering of the elements in the matrix

Matrix Reorderings

- Four reorderings were used:
  1. Natural ordering
     - No changes to the ordering of the elements in the matrix
  2. Reverse Cuthill-McKee
     - Designed to reduce the bandwidth of the matrix

Matrix Reorderings

- Four reorderings were used:
  1. Natural ordering
     - No changes to the ordering of the elements in the matrix
  2. Reverse Cuthill-McKee
     - Designed to reduce the bandwidth of the matrix
  3. Approximate Minimum Degree
     - Designed to reduce the number of non-zeros in the complete factorization for symmetric matrices; observed to have beneficial effects with incomplete LU factorizations for non-symmetric problems[3]

[3] Chow and Patel, "Fine-grained parallel incomplete LU factorization"; Michele Benzi, John C Haws, and Miroslav Tuma. "Preconditioning highly indefinite and nonsymmetric matrices". In: *SIAM Journal on Scientific Computing* 22.4 (2000), pp. 1333–1353.

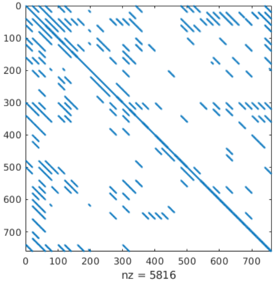
Matrix Reorderings

- Four reorderings were used:
  1. Natural ordering
     - No changes to the ordering of the elements in the matrix
  2. Reverse Cuthill-McKee
     - Designed to reduce the bandwidth of the matrix
  3. Approximate Minimum Degree
     - Designed to reduce the number of non-zeros in the complete factorization for symmetric matrices; observed to have beneficial effects with incomplete LU factorizations for non-symmetric problems[3]
  4. MC64[4]
     - Designed to permute the largest entries in the matrix to the diagonal

[3] Chow and Patel, "Fine-grained parallel incomplete LU factorization"; Benzi, Haws, and Tuma, "Preconditioning highly indefinite and nonsymmetric matrices".

[4] Iain S Duff and Jacko Koster. "On algorithms for permuting large entries to the diagonal of a sparse matrix". In: *SIAM Journal on Matrix Analysis and Applications* 22.4 (2001), pp. 973–996.

Example of the effect of reordering



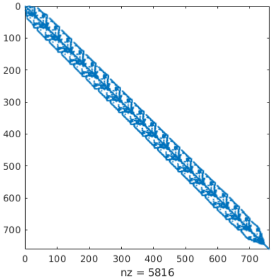


Figure: The sparsity pattern for the 'fs_760_3' matrix with the natural ordering (left), and the RCM ordering (right)

Example of the effect of reordering

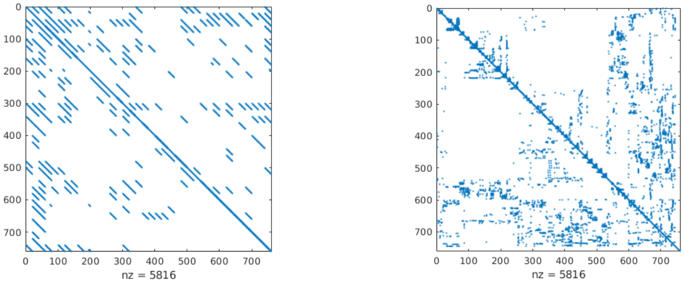


Figure: The sparsity pattern for the 'fs_760_3' matrix with the natural ordering (left), and the AMD ordering (right)

Example of the effect of reordering

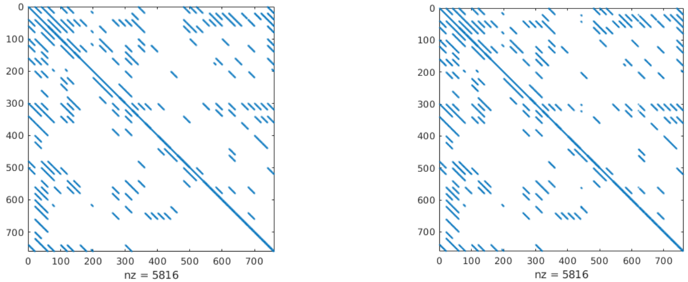


Figure: The sparsity pattern for the 'fs_760_3' matrix with the natural ordering (left), and the MC64 ordering (right)

Increased diagonal dominance

- Diagonal dominance can be increased by using an α-shift[5]. The original matrix, A, can be written,

$$A = D - B \tag{7}$$

where D contains all of the diagonal elements, and B contains all other elements from A.

- Instead of performing the incomplete LU factorization on the original matrix A, the factorization is applied to the matrix,

$$\hat{A} = (1 + \alpha)D - B \tag{8}$$

[5] Thomas A Manteuffel. "An incomplete factorization technique for positive definite linear systems". In: *Mathematics of computation* 34.150 (1980), pp. 473–497.

Effects of increasing α for the OFFSHORE problem

α	FGPILU Sweeps	Krylov solver iterations	Krylov solver time
0	24	30	24.8067
1	9	56	46.4995
10	5	144	130.0958

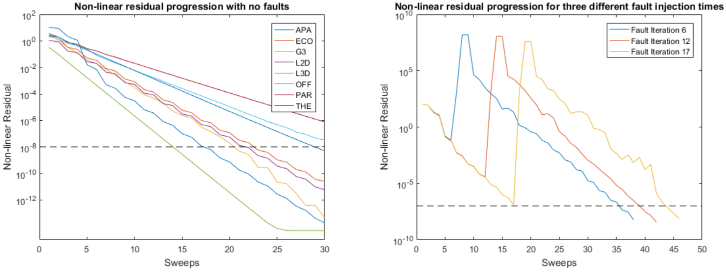
- Making α larger increases the diagonal dominance of the matrix, causing the FGPILU algorithm to converge faster
- Making α larger also increases the difference between A and $\hat{A}$ which potentially lowers the effectiveness of the preconditioner
    - Can be seen by an increased number of Krylov solver iterations and increased Krylov solver execution time

Soft faults

- Soft faults can have an impact on the performance of the FGPILU algorithm by either delaying convergence or preventing it entirely
- The algorithm is naturally resilient to soft faults with less of an impact (e.g. bit flips in less significant bits in the mantissa)
- In this work, faults were modeled in two ways:
  1. As bit flips
     - Reflects a bit flip in unprotected memory
  2. As random perturbations[6]
     - Generalizes the occurrence of a fault to a less specific form of data corruption

[6] Miroslav Stoyanov and Clayton Webster. "Numerical analysis of fixed point algorithms in the presence of hardware faults". In: *SIAM Journal on Scientific Computing* 37.5 (2015), pp. C532–C553; Evan Coleman and Masha Sosonkina. "Evaluating a Persistent Soft Fault Model on Preconditioned Iterative Methods". In: *Proceedings of the 22nd annual International Conference on Parallel and Distributed Processing Techniques and Applications*. 2016.

Potential impact of a fault on the FGPILU algorithm



- Progression of nonlinear residual for 30 sweeps of a typical fault-free run (8 test problems[7]) (left).
- Progression of nonlinear residual for three different fault injection times (right).
- The horizontal dashed line shows a convergence tolerance of 10^{-8}.

[7] Evan Coleman, Masha Sosonkina, and Edmond Chow. "Fault Tolerant Variants of the Fine-Grained Parallel Incomplete LU Factorization". In: *Proceedings of the 2017 Spring Simulation Multiconference*. Society for Computer Simulation International. 2017.

Checkpointing

- Need a mechanism that allows the program to determine if a fault has occurred

Checkpointing

- Need a mechanism that allows the program to determine if a fault has occurred
- Two residuals proposed and used in Chow and Patel[8] and Chow, Anzt, and Dongarra[9] to judge the progression of the fixed point iteration:

[8] Chow and Patel, "Fine-grained parallel incomplete LU factorization".

[9] Edmond Chow, Hartwig Anzt, and Jack Dongarra. "Asynchronous iterative algorithm for computing incomplete factorizations on GPUs". In: *International Conference on High Performance Computing*. Springer. 2015, pp. 1–16.

Checkpointing

- Need a mechanism that allows the program to determine if a fault has occurred
- Two residuals proposed and used in Chow and Patel[8] and Chow, Anzt, and Dongarra[9] to judge the progression of the fixed point iteration:
  - Nonlinear residual:

$$\tau = \sum_{(i,j)\in S} \left| a_{ij} - \sum_{k=1}^{\min(i,j)} l_{ik} u_{kj} \right| \tag{9}$$

[8] Chow and Patel, "Fine-grained parallel incomplete LU factorization".

[9] Chow, Anzt, and Dongarra, "Asynchronous iterative algorithm for computing incomplete factorizations on GPUs".

Checkpointing

- Need a mechanism that allows the program to determine if a fault has occurred
- Two residuals proposed and used in Chow and Patel[8] and Chow, Anzt, and Dongarra[9] to judge the progression of the fixed point iteration:
  - Nonlinear residual:

$$\tau = \sum_{(i,j)\in S} \left| a_{ij} - \sum_{k=1}^{\min(i,j)} l_{ik} u_{kj} \right| \tag{9}$$

  - ILU residual:

$$||A - LU||_F \tag{10}$$

[8] Chow and Patel, "Fine-grained parallel incomplete LU factorization".

[9] Chow, Anzt, and Dongarra, "Asynchronous iterative algorithm for computing incomplete factorizations on GPUs".

Typical progression of residual norms

ecl-ILU	ecl-NL	off-ILU	off-NL
3.38E+04	3.13E+04	5.16E+01	2.81E+01
2.44E+04	9.44E+03	4.60E+01	1.36E+01
2.72E+04	3.74E+03	4.43E+01	6.02E+00
2.90E+04	1.41E+03	4.40E+01	2.77E+00
2.97E+04	4.96E+02	4.39E+01	1.31E+00
2.99E+04	1.74E+02	4.39E+01	6.46E-01
3.00E+04	6.41E+01	4.39E+01	3.20E-01
3.00E+04	2.14E+01	4.39E+01	1.64E-01
3.00E+04	6.99E+00	4.39E+01	8.41E-02
3.00E+04	3.99E+00	4.39E+01	4.03E-02

- Note that the nonlinear residual will continue to decrease until convergence of the FGPILU algorithm is reached, but the ILU residual quickly settles to a particular value
- Problems represent one symmetric ('off') and one non-symmetric ('ecl') matrix used in this study

Checkpointing

- Obvious idea: Monitor the progression of the nonlinear residual norm, and declare a fault if $\tau^{k+r} > \gamma \cdot \tau^{k}$
- Solution: If there is a fault, roll-back the entire factor(s) (either L or U) to the last known good state
- Parameters:
  - γ: how strict to make the check
  - r: how often to make the check

Outline

1. Overview
   - Fine-Grained Parallel Incomplete LU Factorization
2. Convergence and Resilience
   - Examining the Convergence of the FGPILU Algorithm
   - Soft Fault Resilience
3. Numerical Results
   - Set Up
   - Convergence Results
   - Resilience Results
4. Future Directions

Experiment Set-Up

- Experiments were conducted on the Turing High Performance Cluster at Old Dominion University using a single Tesla K80 GPU
- Made use of the MAGMA library for: input/output routines, linear solvers, and the unmodified FGPILU routine
- Transient soft faults were injected on a single sweep of the fixed point iteration; results were averaged over multiple runs
- 2 non-symmetric matrices were used along with a single SPD matrix for extended analysis
    - The two non-symmetric matrices come from previous work on iterative methods for non-symmetric problems[10]
    - The single SPD matrix has the highest conditioning number of matrices previous studies[11]

[10] Edmond Chow and Yousef Saad. “Experimental study of ILU preconditioners for indefinite matrices”. In: *Journal of Computational and Applied Mathematics* 86.2 (1997), pp. 387–414; Xiaoye S Li and James Demmel. “A Scalable Sparse Direct Solver Using Static Pivoting.”. In: *PPSC*. 1999; Anshul Gupta. “Improved symbolic and numerical factorization algorithms for unsymmetric sparse matrices”. In: *SIAM Journal on Matrix Analysis and Applications* 24.2 (2002), pp. 529–552.

[11] Chow, Anzt, and Dongarra, “Asynchronous iterative algorithm for computing incomplete factorizations on GPUs”; Coleman, Sosonkina, and Chow, “Fault Tolerant Variants of the Fine-Grained Parallel Incomplete LU Factorization”; Evan Coleman and Masha Sosonkina. “Self-Stabilizing Fine-Grained Parallel Incomplete LU

Experiment Set-Up

- Impacts on the preparation of the preconditioner (e.g. the FGPILU algorithm itself) and on the use of the preconditioner in a linear solver were studied
- Note: to fully judge the impact of transient faults, the fixed point iteration in the FGPILU algorithm was run until the nonlinear residual norm was excessively small
    - Allows for a more complete look at the performance of the algorithm with respect to soft faults, but artifically inflates timing results
- The initial guess, x_0, was set to be a zero vector in all cases
- The initial guess for the L and U factors was set to be the components of A in the same location

Summary of the matrices used

Matrix Name	SPD?	COND EST	Dim.	Non-zeros	Description
fs_760_3	N	9.93E+19	760	5,816	chemical engineering
ecl32	N	9.41E+15	51,993	380,415	circuit simulation
OFF SHORE	Y	2.24E+13	259,789	4,242,673	transient electric field diffusion

- Many other matrices from[12] were experimented with
  - Most other matrices did not converge for a satisfactory number of permutations of α and level of ILU fill-in with the standard initial guess
- To help improve convergence, all problems were scaled to have unit diagonal

[12] Chow and Saad, "Experimental study of ILU preconditioners for indefinite matrices"; Benzi, Haws, and Tuma, "Preconditioning highly indefinite and nonsymmetric matrices".

Experiment Combinations

- For each of the three matrices that were tested:
  - four orderings were tested (`MC64`, `AMD`, `RCM`, and the natural ordering),
  - 3 level of ILU fill-in were tested (levels 0, 1, and 2),
  - and 3 factors for α were used (0, 0.5, and 1.0).
- This leads to a total of 108 permutations:
  - 84 (77.78%) led to a case were the FGPILU algorithm converged
  - Only 29 (26.85%) resulted in a successful GMRES solve
  - Details on the parameters that led to these successful solves are provided on the next slide

Detailed Experiment Results

Matrix	Ordering	α	ILU Level	Sweeps	Krylov Its.	Time (s)
offshore	AMD	0	0	19	30	18
offshore	AMD	0.5	0,1,2	10,11,11	40,34,34	24,55,144
offshore	AMD	1	0,1,2	8,9,9	56,54,54	34,96,229
offshore	RCM	0	0	19	19	35
offshore	RCM	0.5	0,1,2	10,11,11	37,34,34	68,306,771
offshore	RCM	1	0,1,2	9,9,9	54,54,54	101,484,1226
offshore	Natural	0	0	22	22	84
offshore	Natural	0.5	0,1,2	11,12,12	38,34,34	146,312,695
offshore	Natural	1	0,1,2	9,10,10	54,54,54	210,491,1104
ecl32	AMD	0	2	15	127	104
ecl32	RCM	0	2	24	9	39
ecl32	Natural	0	2	18	11	16
fs_760_3	AMD	0	2	55	3	0.4
fs_760_3	RCM	0	1,2	52,63	2,2	0.4,0.4
fs_760_3	MC64	0	1	16	3	0.3
fs_760_3	Natural	0	1	16	3	0.3

Experiment Observations

- The two non-symmetric problems tend to perform better with smaller values of α and higher levels of fill-in allowed
- The level of ILU fill-in tends to not have as much of an impact on whether or not the problem can be solved when compared to the ordering or value for α, but affects the performance
- In the results found here, the benefit of having more complete L and U factors from going to a higher fill-in level tends to be outweighed by the increased computational cost of the fixed point iteration associated with the FGPILU algorithm for a drastically larger number of elements.
  - Increased parallelism is possible with more elements which may be able to be better leveraged by future hardware

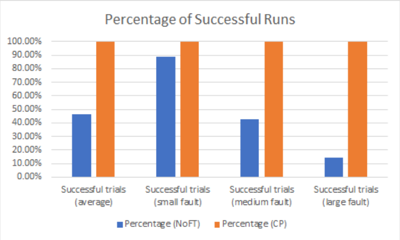
Matrix	nnz(ILU-0)	nnz(ILU-1)	nnz(ILU-2)
offshore	4.5mil	10.0mil	21.7mil
ecl32	0.4mil	1.0mil	2.0mil
fs_760_3	6.6k	17.6k	32.3k

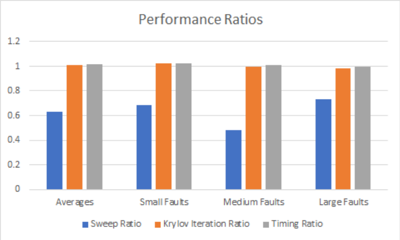
Soft fault resilience

- Given that the fixed point iteration does not converge for every combination of parameters, it is natural to wonder how a soft fault may affect the convergence
- In the symmetric case, several strategies have been tested for soft fault resilience[13]
- Only checkpointing based on the progression of the nonlinear residual norm was tested for these matrices since it was the most robust variant of the FGPILU algorithm developed for symmetric matrices

[13] Evan Coleman and Masha Sosonkina. "Fault Tolerance for Fine-Grained Iterative Methods". In: *Proceedings of the 7th annual Virginia Modeling, Simulation, and Analysis Center Capstone Conference*. Virginia Modeling, Simulation, and Analysis Center. 2017; Coleman and Sosonkina, "Self-Stabilizing Fine-Grained Parallel Incomplete LU Factorization".

Soft fault resilience





- Only the 29 successful cases were tested in this section
- Checkpointing was always able to restore convergence
- Checkpointing added minimally to the number of Krylov iterations and total time, but came at the cost of several extra sweeps of the FGPILU algorithm

Outline

1. Overview
   - Fine-Grained Parallel Incomplete LU Factorization
2. Convergence and Resilience
   - Examining the Convergence of the FGPILU Algorithm
   - Soft Fault Resilience
3. Numerical Results
   - Set Up
   - Convergence Results
   - Resilience Results
4. Future Directions

Future Directions

- This talk showcased several potential strategies for increasing the convergence rate of the FGPILU algorithm
- Several main directions for how to move forward with this work:
  1. Continue experimenting with other techniques for increasing the convergence rate
  2. Extend the knowledge gained about convergence of the asynchronous FGPILU algorithm to a less specific setting of general asynchronous iterative methods
  3. Expand the test suite of problems to encapsulate more matrices / domain areas
  4. Test other resilience methods for this (or any other) more difficult problem set

Future Directions

- Other potential applications for the FGPILU fixed point iteration:
  - More complicated preconditioning routines that make use of fixed point sweeps
  - Fine-grained generation and application of preconditioner in conjunction with fine-grained sparse triangular solves

Questions?