Fault Tolerant Variants of the Fine-Grained Parallel Incomplete LU Factorization

Evan Coleman
NSWC - Dahlgren Division
Old Dominion University
ecole028@odu.edu

Masha Sosonkina
Old Dominion University
msosonki@odu.edu

Edmond Chow
Georgia Institute of Technology
echow@cc.gatech.edu

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

1. Introduction to the Fine-Grained Parallel Incomplete LU factorization
2. Techniques for fault tolerance
3. Results
4. Future directions

Outline

1. Introduction to the Fine-Grained Parallel Incomplete LU factorization
2. Techniques for fault tolerance
3. Results
4. Future directions

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
- Focus area: Iterative methods in linear algebra
- Outline 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

Incomplete LU factorization

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

$$A \approx LU$$

- Define the sparsity pattern as,

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

- Chow and Patel* make the observation that,

$$(LU)_{ij} = a_{ij}$$

for $(i,j) \in S$

*Chow, E., and A. Patel. 2015. "Fine-grained parallel incomplete LU factorization". SIAM Journal on Scientific Computing vol. 37 (2), pp. C169–C193.

Incomplete LU 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
- Make use of the constraint,

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

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

Fine-Grained Parallel Incomplete LU Factorization

- Leads to two non-linear equations
  1. $l_{ij} = \frac{1}{u_{jj}}\left(a_{ij} - \sum_{k=1}^{j-1} l_{ij}u_{kj}\right)$
  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,

$$x^{k+1} = G\left(x^k\right)$$

  where G captures the two equations above and an initial guess x^0 is supplied
- Higher degree of parallelism: allows one thread to be assigned to update each component

Outline

1. Introduction to the Fine-Grained Parallel Incomplete LU factorization
2. Techniques for fault tolerance
3. Results
4. Future directions

Techniques

- Three techniques investigated
  - Checkpointing
  - Partial checkpointing
  - Self-stabilizing periodic correction step

Checkpointing

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

$$\tau = ||(A - LU)_S||_F = \left[\sum_{(i,j)\in S} \left(a_{ij} - \sum_{k=1}^{\min(i,j)} l_{ik}u_{kj} \right)^2 \right]^{\frac{1}{2}}$$

  - ILU residual

$$||A - LU||_F$$

*Chow, E., and A. Patel. 2015. "Fine-grained parallel incomplete LU factorization". SIAM Journal on Scientific Computing vol. 37 (2), pp. C169–C193.

**Chow, E., H. Anzt, and J. Dongarra. 2015. "Asynchronous iterative algorithm for computing incomplete factorizations on GPUs". In International Conference on High Performance Computing, pp. 1–16. Springer.

Checkpointing

- Typical progression – Apache2

Iteration (k)	Non-linear residual (τ)	ILU Residual
1	1.05e+02	379.88
2	8.81e+01	376.74
3	2.38e+01	367.10
4	1.36e+01	366.70
5	2.39e+00	366.45
6	1.21e+00	366.45
7	5.24e-01	366.45
8	2.24e-02	366.45
9	1.05e-03	366.45

Checkpointing

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

Partial Checkpointing

- Motivating goal: avoid rolling back the entire computed factors
- Idea: monitor the individual components, τ_{ij}, of the non-linear residual norm

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

- The individual non-linear residual norms are generally decreasing
  - Examining component wise progression shows the progression is not monotonic
  - To limit the number of false positive a check on the trend of the global non-linear residual norm, $\frac{d\tau}{dt}$, is added

Partial Checkpointing

- If a fault is detected the number of components that are rolled back is limited
- The individual non-linear norm computation,

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

  corresponds to portions of one row of L and one column of U
  - The entirety of the affected row and column are rolled back if a fault is detected
- Similar parameters to the first checkpointing scheme exist to determine the frequency and sharpness of the fault detection mechanism

Self-Stabilizing

- Sao and Vuduc* proposed a self-stabilizing variant of the Conjugate Gradient algorithm that uses a periodic correction step

- Principles:
  - System will enter a valid state (no matter the initial state) in a finite number of steps
  - Uses the periodic correction step to restore sufficient conditions for convergence
    - Eliminates the need for explicit fault detection

*Sao, P., and R. Vuduc. 2013. "Self-stabilizing iterative solvers". In Proceedings of the Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, pp. 4. ACM.

Self-Stabilizing

- Investigated the use of a periodic correction step to make the FGPILU algorithm resilient to transient soft faults
- In order to develop a periodic correction step (with no explicit fault detection) the performance of the FGPILU on the two dimensional discretization of the Laplacian was examined
  - In particular:
    - Progression of the individual components
    - Progression of the individual non-linear residual norms, τ_{ij}
    - Progression of the global non-linear residual norm, τ

Self-Stabilizing

- Developed a periodic correction step:
  - Fine-grained
  - No explicit error detection
  - No communication needed between threads
- Based on checking:
  - Size of the current component
  - Relative change in the current component
- Note: does not generalize to all other problems
  - Convergence through faults is not guaranteed
    - Depends on the structure of the domain and the progression of the norm of the Jacobian

Outline

1. Introduction to the Fine-Grained Parallel Incomplete LU factorization
2. Techniques for fault tolerance
3. Results
4. Future directions

Experiment set up

- Hardware/software set up:
  - Turing HPC cluster at Old Dominion University
    - Used a single Nvidia K40m Tesla GPU
  - Made use of the MAGMA library for:
    - Input/output routines
    - Initial FGPILU implementation
    - Linear solvers
- Problems
  - 2D and 3D discretizations of the Laplacian
  - 6 other problems from the University of Florida sparse matrix collection
    - (same set of problems used in Chow, Anzt, and Dongarra*)

*Chow, E., H. Anzt, and J. Dongarra. 2015. "Asynchronous iterative algorithm for computing incomplete factorizations on GPUs". In International Conference on High Performance Computing, pp. 1–16. Springer.

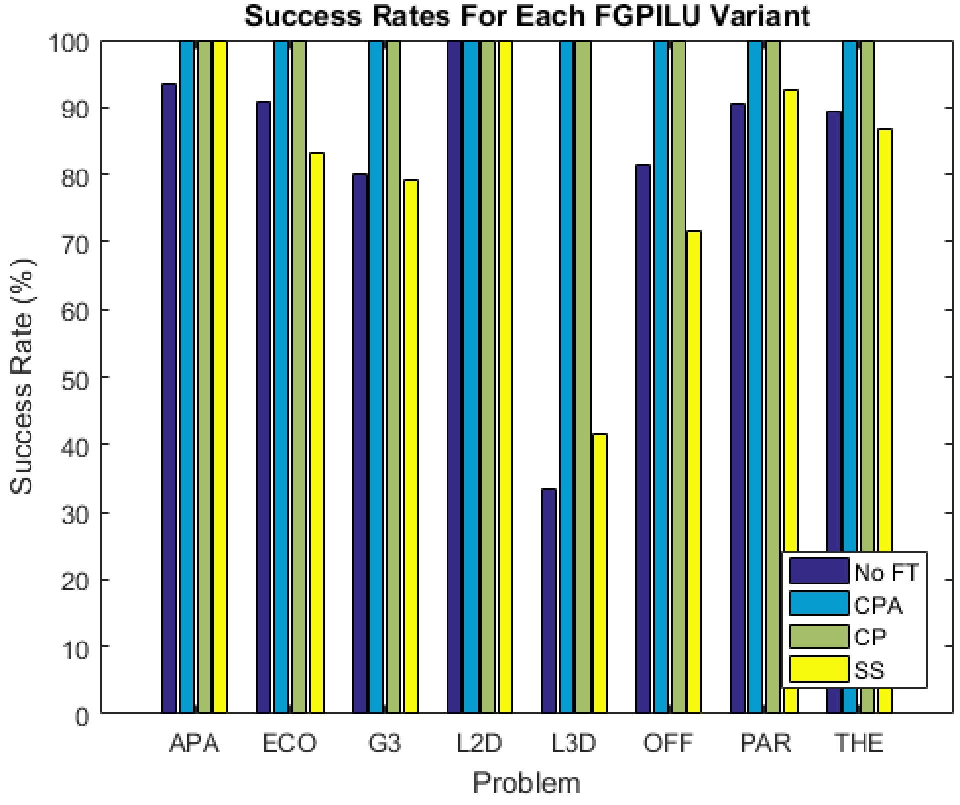
Experiment set up

- To help improve convergence all problems were
  - Re-ordered (Reverse Cuthill-Mckee)
  - Scaled to have unit diagonal
- Transient soft faults injected using a perturbation-based methodology*
  - Faults were injected on a single iteration of the fixed-point iteration to generate the incomplete LU factors
  - Results were averaged over multiple runs
- Impacts on the preparation of the preconditioner and the effect of using the resultant preconditioner were studied
- Note: to fully judge the impact of transient faults, the fixed-point iteration in the FGPILU algorithm was run until the non-linear residual norm was excessively small
  - Allows for a more complete look at the performance of the algorithm with respect to soft faults
  - Artificially inflates timing results relative to traditional incomplete factorizations

Coleman, E., and M. Sosonkina. 2016. "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.

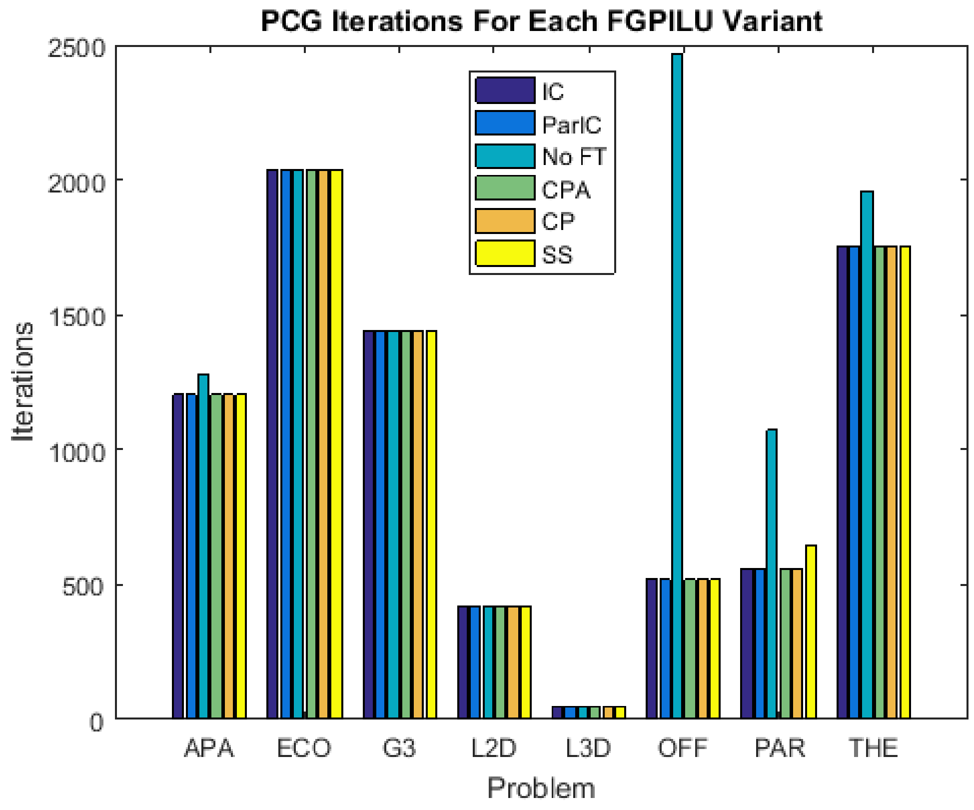
Results

- Success corresponds to a successful solve of the linear system
- Both checkpointing variants seem to be resilient to transient soft faults
- The self-stabilizing method works well for the problem it was designed for, but breaks down in the general case



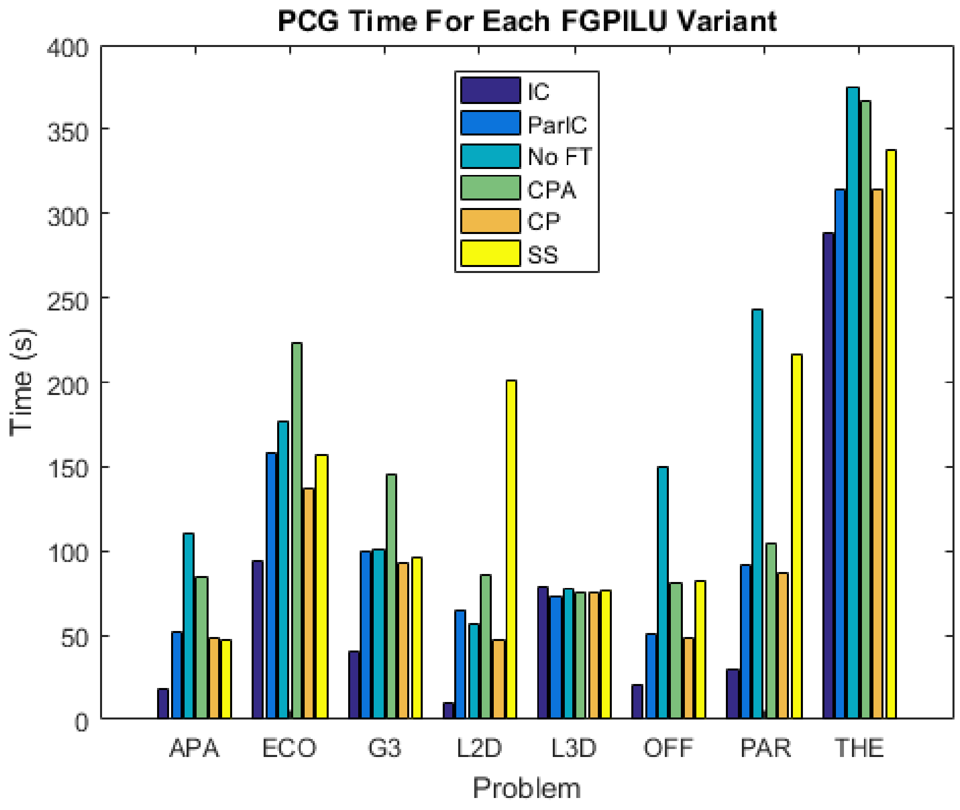
Results

- *Number of iterations* for the linear solver to converge using incomplete LU factors from the different variants discussed as a preconditioner
- Note: only judged across "successful" runs



Results

- *Time (s)* for the linear solver to converge (including preconditioner preparation) using incomplete LU factors from the different variants discussed as a preconditioner
- Note: only judged across "successful" runs



Outline

1. Introduction to the Fine-Grained Parallel Incomplete LU factorization
2. Techniques for fault tolerance
3. Results
4. Future directions

Summary and Future Directions

- This work:
  - Presented some initial results showing possible strategies for fault tolerance of the FGPILU algorithm
- In the future:
  - Improve the performance of the developed techniques
  - Expand on the self-stabilizing approach
  - Apply the developed techniques to other fine-grained methods
  - Work at generalizing results to a broader setting

Questions?

References

- Asanovic, K., R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson,W. Plishker, J. Shalf, S.Williams et al. 2006. “The landscape of parallel computing research: A view from Berkeley”. Technical report, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley.
- Bridges, P., K. Ferreira, M. Heroux, and M. Hoemmen. 2012. “Fault-tolerant linear solvers via selective reliability”. arXiv preprint arXiv:1206.1390.
- Bronevetsky, G., and B. de Supinski. 2008. “Soft error vulnerability of iterative linear algebra methods”. In Proceedings of the 22nd annual international conference on Supercomputing, pp. 155–164. ACM.
- Cappello, F., A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir. 2014. “Toward exascale resilience: 2014 update”. Supercomputing frontiers and innovations vol. 1 (1).
- Chow, E., H. Anzt, and J. Dongarra. 2015. “Asynchronous iterative algorithm for computing incomplete factorizations on GPUs”. In International Conference on High Performance Computing, pp. 1–16. Springer.
- Chow, E., and A. Patel. 2015. “Fine-grained parallel incomplete LU factorization”. SIAM Journal on Scientific Computing vol. 37 (2), pp. C169–C193.
- Coleman, E., and M. Sosonkina. 2016a. “A Comparison and Analysis of Soft-Fault Error Models using FGMRES”. In Proceedings of the 6th annual Virginia Modeling, Simulation, and Analysis Center Capstone Conference. Virginia Modeling, Simulation, and Analysis Center.
- Coleman, E., and M. Sosonkina. 2016b. “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.
- Davis, TA 1994. “The University of Florida Sparse Matrix Collection”. http://www.cise.ufl.edu/research/sparse/matrices/
- Elliott, J., M. Hoemmen, and F. Mueller. 2015. “A Numerical Soft Fault Model for Iterative Linear Solvers”. In Proceedings of the 24nd International Symposium on High-Performance Parallel and Distributed Computing.
- Frommer, A., and D. Szyld. 2000. “On asynchronous iterations”. Journal of computational and applied mathematics vol. 123 (1), pp. 201–216.
- Geist, A., and R. Lucas. 2009. “Major computer science challenges at exascale”. International Journal of High Performance Computing Applications.
- Innovative Computing Lab 2015. “Software distribution of MAGMA”. http://icl.cs.utk.edu/magma/.
- Saad, Y. 2003. Iterative methods for sparse linear systems. Siam.
- Sao, P., and R. Vuduc. 2013. “Self-stabilizing iterative solvers”. In Proceedings of the Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, pp. 4. ACM.
- Snir, M., R.Wisniewski, J. Abraham, S. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson et al. 2014. “Addressing failures in exascale computing”. International Journal of High Performance Computing Applications.