Impacts of Three Soft-Fault Models on Hybrid Parallel Asynchronous Jacobi

Evan Coleman, Erik Jensen and Masha Sosonkina
Old Dominion University

Sparse Days, September 27–28, 2018 CERFACS, Toulouse, France

Acknowledgments

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

Introduction

- The prevalence of faults is expected to increase as platforms continue to grow
- This calls for the design of fault-tolerant computational algorithms that are robust in the face of these failures
- Development of such algorithms has become one of many priorities on the road towards exascale
- At a high level, computing faults can be divided into two distinct categories: hard faults and soft faults
  - Soft faults do not cause immediate program interruption and are the focus of this work

Motivation

- Soft faults typically manifest as bit flips currently, but the exact rate and manner soft faults will manifest on future HPC platforms is not known
- Traditionally studies on soft faults rely only on running a large number of trials injecting a random bit flip and seeing how an algorithm responds
  - This tends to reveal average case behavior
- Using numerical fault simulation techniques allows the size and frequency of faults to be controlled
- This study judges the impact of soft faults as injected by random bit flips against the impact of two different numerical soft fault models

Asynchronous Iterative Methods

- Asynchronous iterative methods represent a broad class of algorithms whereby, in the course of the iterative updates, different components may be updated at different times
- These algorithms are popular for stationary iterative solvers, i.e. iterative processes that take the form,

$$x^{(k+1)} = Bx^{(k)} + c \tag{1}$$

  for some iteration matrix B
- These methods can help mitigate the cost of synchronization by allowing each component, x_l, to be updated without waiting for the latest information regarding other components

Asynchronous Jacobi

- Jacobi traditionally splits the input matrix A into the diagonal, D, and lower and upper triangular portions, L and U
- The update can then be written,

$$x^{(k+1)} = D^{-1}(L + U)x^{(k)} + D^{-1}b \tag{2}$$

- Alternatively, for finite-difference discretizations of partial differential equations on regular grids, the update can be performed on the grid directly
- Example: For the 2D Laplacian, this causes each update equation to become

$$v_{l,m}^{k+1} = \frac{1}{4}\left(v_{l+1,m}^{k} + v_{l-1,m}^{k} + v_{l,m+1}^{k} + v_{l,m-1}^{k}\right) \tag{3}$$

Hybrid Parallelism

- In this study, the asynchronous Jacobi algorithm was implemented in a hybrid parallel manner using both MPI and OpenMP
  - Several previous studies have analyzed asynchronous Jacobi[1]
- The solution to a discretization of a PDE on an $n \times n$ grid is sought
  - Including boundary conditions, the grid becomes $(n+2) \times (n+2)$
- Given $P+1$ MPI processes, the domain is divided into equal subdomains among P of the MPI processes
  - One MPI process is reserved to aid in communication, memory transfer, and global calculations

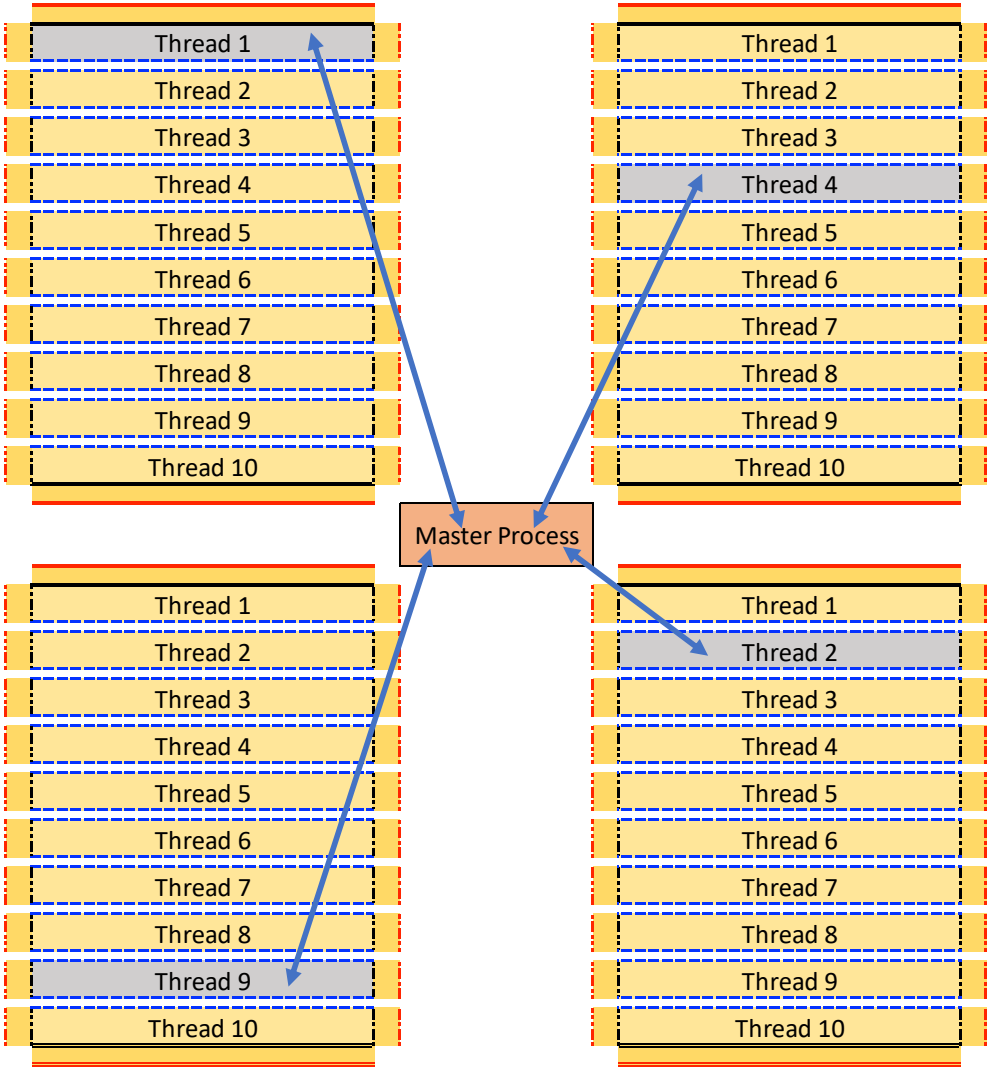
[1] James Hook and Nicholas Dingle. "Performance analysis of asynchronous parallel Jacobi". In: *Numerical Algorithms* (2013), pp. 1–36; Iain Bethune et al. "Performance analysis of asynchronous Jacobi's method implemented in MPI, SHMEM and OpenMP". In: *The International Journal of High Performance Computing Applications* 28.1 (2014), pp. 97–111.

Hybrid Parallelism (cont'd)

- Given n_p OpenMP threads, the work is equally divided among the n_p threads
- Each thread solves for n^2/n_p grid points
- The grid that the PDE is discretized over is evenly partitioned along the y-axis
  - e.g., each thread is assigned a given number of partial rows of the grid
- Internally, two matrices U_0 and U_1 store the grid point values that each thread reads
  - e.g. from U_1, to compute newer values to write to U_0
  - As the method is asynchronous, each thread independently determines which matrix stores its newer vs older values
- OpenMP locks are employed to ensure that boundary values are captured accurately, i.e. that multiple threads do not attempt to make simultaneous updates

Diagram of Hybrid Parallel Implementation

Per process, thread is assigned to communicate in a round-robin manner



Recovery Techniques

- A single checkpointing variant of the asynchronous Jacobi algorithm is used here to help the algorithm converge despite the occurrence of a fault
- Traditional checkpointing typically relies upon monitoring the progression of the residual,

$$r^{(k)} = b - Ax^{(k)} \tag{4}$$

from one iteration, $x^{(k)}$, to the next, $x^{(k+1)}$, and rolling back the affected components

Checkpointing

- Detection: Monitor the progression of the residual, and declare a fault if $r^{(k)} > \gamma \cdot r^{(k+d)}$
- Solution: If there is a fault, roll-back the entire vector x to the last known good state
- Parameters:
  - γ: how strict to make the check
  - d: how often to make the check

Modified Checkpointing

- Checkpointing in a global manner such as this works against the fine-grained nature of the asynchronous Jacobi algorithm
- Solution: Employ partial checkpointing[2] where each OpenMP thread monitors the progression of its local contribution to the residual
  - i.e. monitor the progression of $r_l^{(k)}$ for each thread l, and roll back only the affected components
- This causes the checkpointing equation to become

$$r_l^{(k)} > \gamma \cdot r_l^{(k+d)}, \tag{5}$$

aimimg to preserve the fine-grained nature of the algorithm

[2] Evan Coleman and Masha Sosonkina. "Self-Stabilizing Fine-Grained Parallel Incomplete LU Factorization". In: *Sustainable Computing: Informatics and Systems* (2018).

Modified Checkpointing (cont'd)

- Difficulties:
  - Determining which components are "affected" is hard since the effect of the faults propagate from one component to another when communication occurs
    - After some initial experiments, here, all components assigned to a given MPI process are reset if a fault is detected by any thread in that subdomain
    - Experimenting with various definitions and communication strategies is left as future work
  - Progression of the residuals on a thread-by-thread level is not always monotonic
    - Setting γ too close to 1 can result in an excess of false positives

Soft Fault Models

- Three methods for injecting soft faults are used:
  1. Direct injection of bit flips
     - Randomly selected across all bits
  2. Perturbation-based Soft Fault Model (PBSFM)[3]
     - Injects a perturbation into all components in the affected subdomain
     - Perturbations are sampled uniformly from a user selected range
     - Size of the perturbation corresponds to the impact of the fault
  3. Shuffled-based Soft Fault Model (SBSFM)[4]
     - Shuffles all components of the affected subdomain
     - Scales the result by a factor α
     - Size of α corresponds to the impact of the fault

---

[3] 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.

[4] James Elliott, Mark Hoemmen, and Frank Mueller. "A Numerical Soft Fault Model for Iterative Linear Solvers". In: *Proceedings of the 24nd International Symposium on High-Performance Parallel and Distributed Computing*. 2015.

Comparison of Soft Fault Models

- A quick comparison of the fault models can help illustrate some of the differences between them
- A simple MATLAB experiment was conducted where a vector x had a fault injected by all three methods, and x was taken to be the 50th iterate generated by synchronous Jacobi in the solution of the 2D Laplacian over a 400×400 grid with a starting vector of all zeros
- Three values of perturbation were used for the PBSFM
  - large: $\tau_i \in (10^{-16}, 10^{16})$, medium: $\tau_i \in (10^{-8}, 10^{8})$, and small: $\tau_i \in (10^{-2}, 10^{2})$
- Three values of scaling factor for SBSFM
  - large: $\alpha = 10^{14}$, medium: $\alpha = 10^{6}$, and small: $\alpha = 10^{2}$

Comparison of Soft Fault Models (Over 10,000 trials)

Analysis of the total amount c of data corruption:

	Mean(c)	Median (c)	Mean(log(c))	Std(log(c))
PBSFM (L)	3.65E+17	3.65E+17	40.44	0.007
PBSFM (M)	3.65E+09	3.65E+09	22.02	0.007
PBSFM (S)	3.65E+03	3.65E+03	8.20	0.007
SBSFM (L)	5.77E+17	5.77E+17	40.90	7.30E-12
SBSFM (M)	5.77E+09	5.77E+09	22.48	7.41E-09
SBSFM (S)	5.74E+05	5.74E+05	13.26	7.51E-05
BF	1.13E+304	3.05E-05	-0.81	52.8

Experiment setup

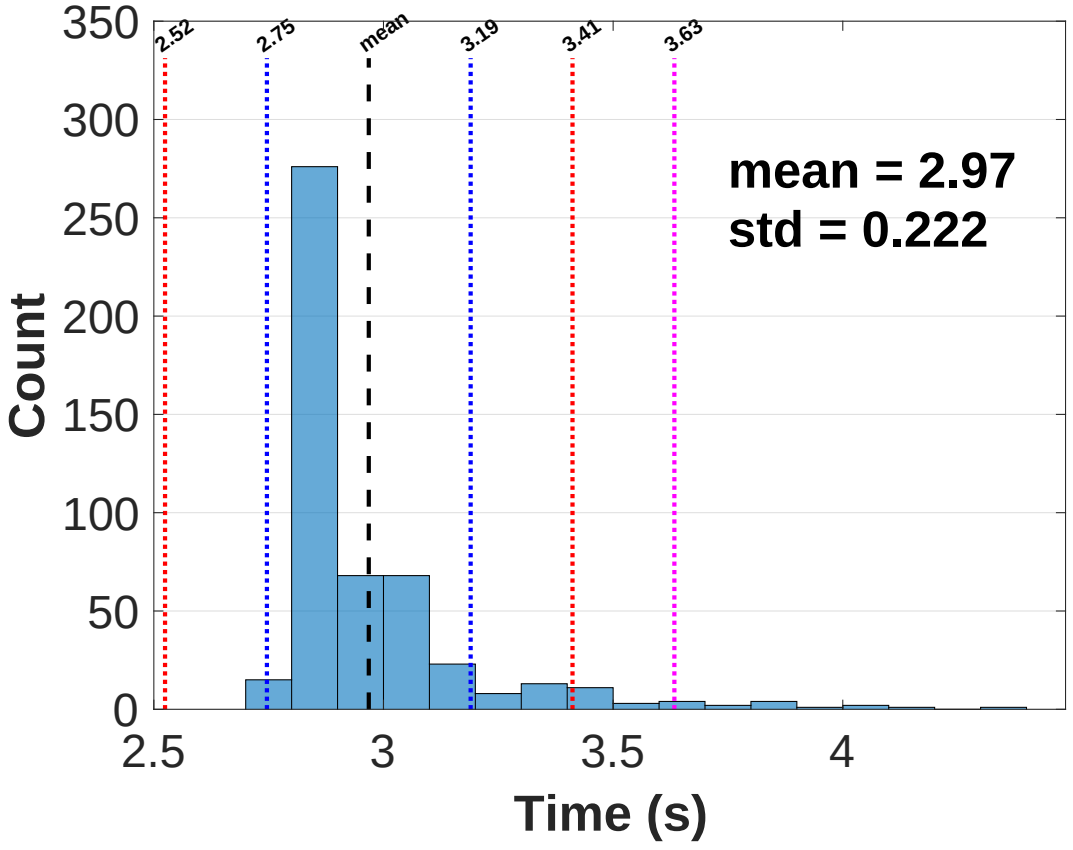
- The series of hybrid parallel experiments were conducted on the Turing High Performance Computing cluster at Old Dominion University
  - This cluster contains 190 standard compute nodes, 10 GPU nodes, 10 Intel Xeon Phi Knight's Corner nodes, and 4 high memory nodes, connected with a Fourteen Data Rate (FDR) InfiniBand network
- Compute nodes contain 16–32 cores and 128 GB of RAM
- Data for these experiments was collected on sockets consisting of 10 Intel Xeon E5-2670 (Haswell) v2 2.50 Ghz cores

Problem description

- The problem solved in all experiments is a 2D finite-difference discretization of the Laplacian
- Hybrid parallel implementation as described earlier:
  - Taken over a 400×400 grid
  - 5 MPI processes used (1 reserved for communication)
  - Subdomains of size 200×200 assigned to each of the 4 main MPI processes
  - Each MPI process divided by row among 10 OpenMP threads
  - Each thread is then responsible for 200×20 components to update

Baseline results

- Before discussing results concerning soft faults, statistics for fault-free solves should be discussed
- Sample problem was solved 500 times without faults



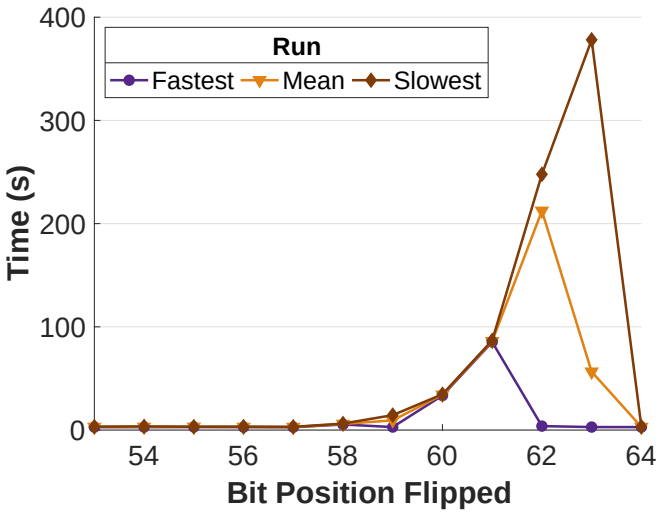
Impact of Soft Faults

- This series of experiments shows the impact of soft faults
- All three soft fault models were used:
  - Perturbation Based Soft Fault Model (PBSFM)
    - Perturbation values were taken from $(10^{-2j}, 10^{2j})$ for $j = 1, \ldots, 8$.
  - Shuffle Based Soft Fault Model (SBSFM)
    - The scaling factors used were $10^2, 10^6, 10^{10}, 10^{16}$.
  - Bit Flip Soft Fault Model (BFSFM)
    - Bits were selected in a uniform random manner

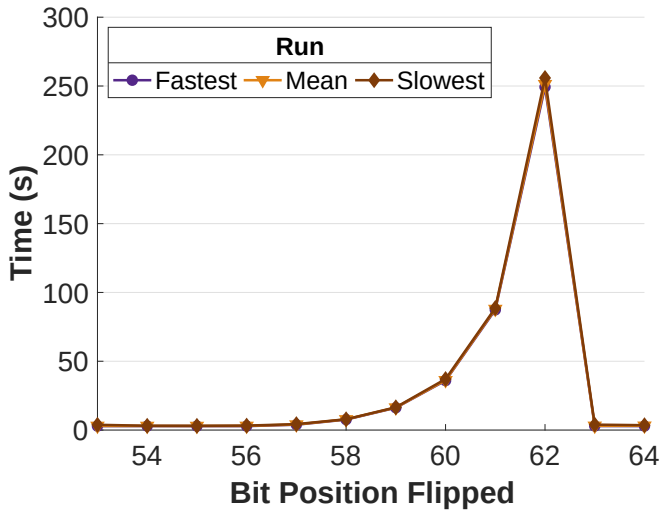
Impact of Soft Faults (cont'd)

- Based on the mean runtime of 2.97s, three different fault injection times were used:
  - $Early = 0.1s$
  - $Middle = 1.2s$
  - $Late = 2.5s$
- Each experiment was replicated 7 times
  - The average of the seven runs, along with the fastest and slowest are shown in all figures

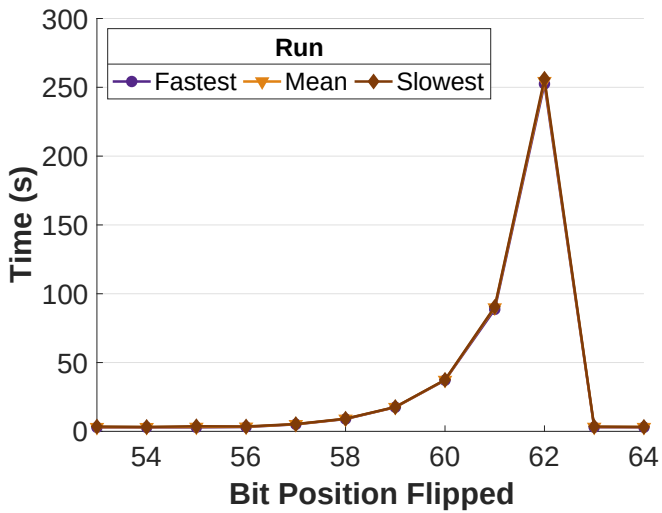
Bit Flips (exponent and sign bits only)



(a) Early

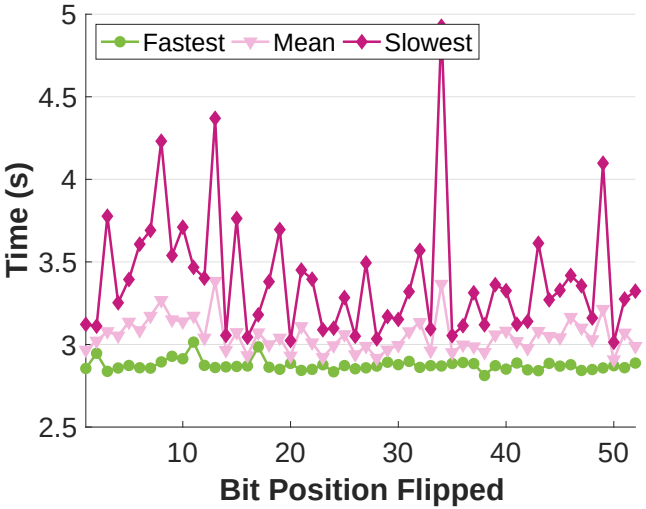


(b) Middle

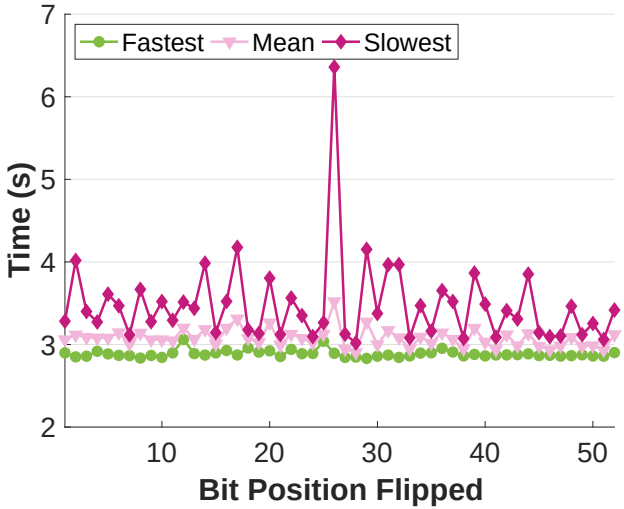


(c) Late

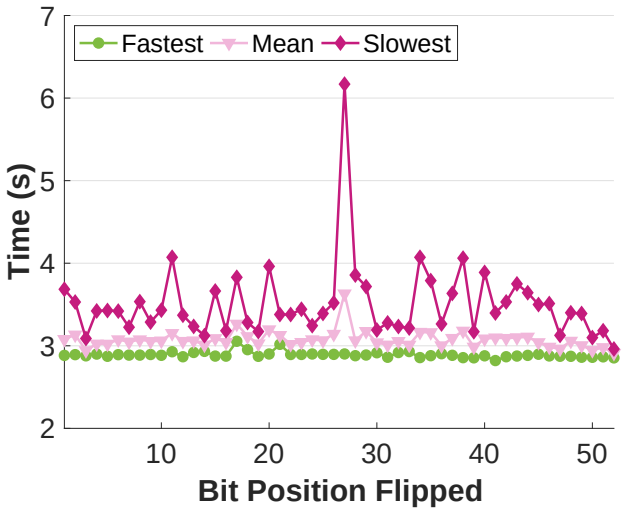
Bit Flips (mantissa)



(a) Early

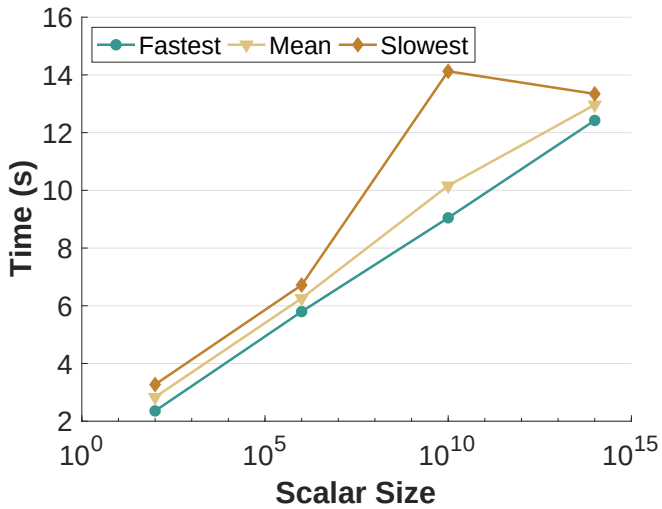


(b) Middle

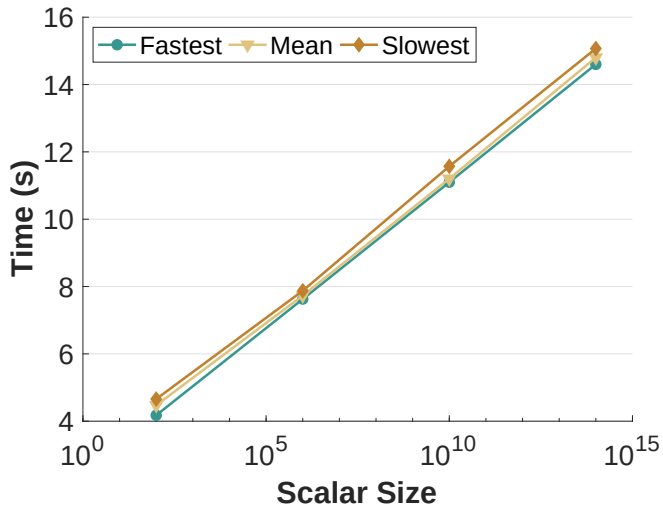


(c) Late

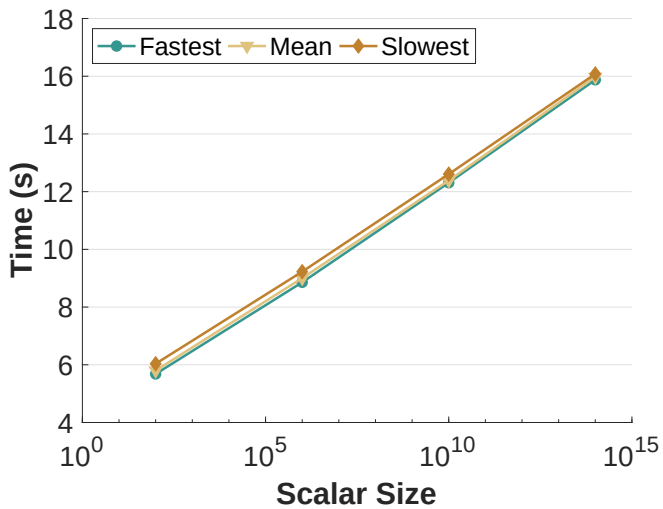
SBSFM



(a) Early

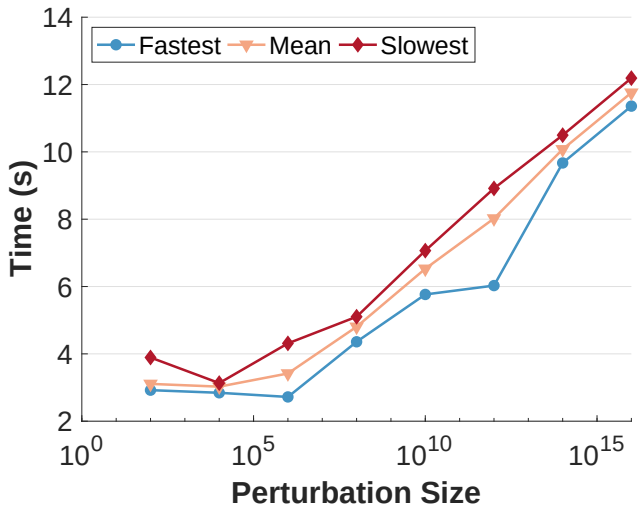


(b) Middle

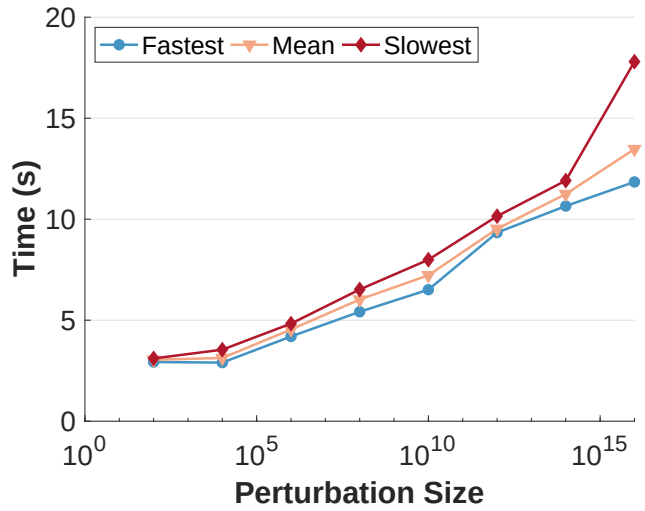


(c) Late

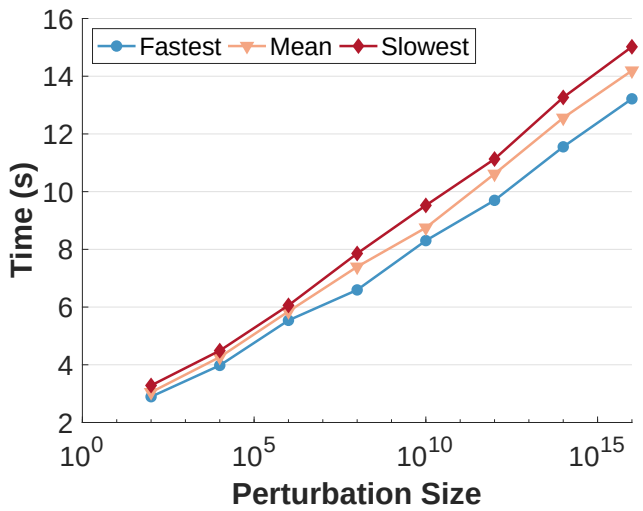
PBSFM



(a) Early



(b) Middle

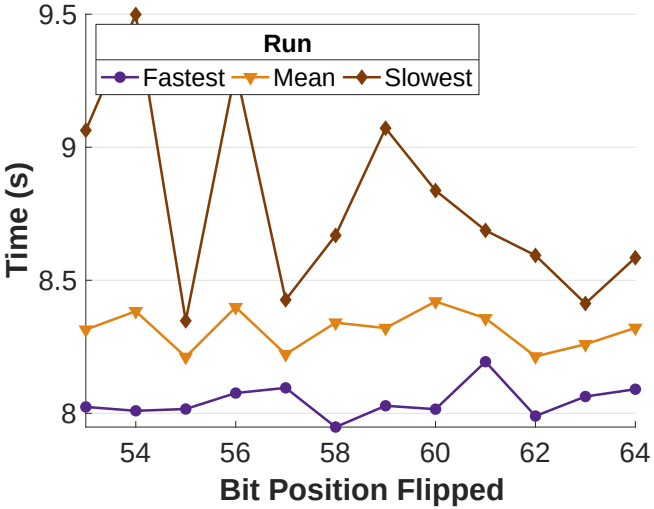


(c) Late

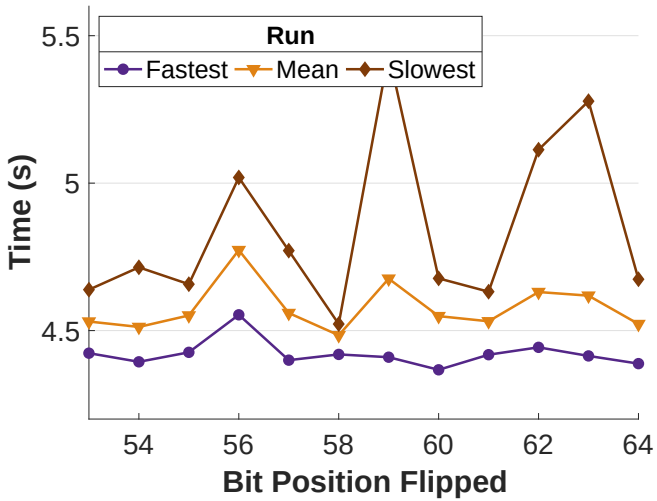
Recovery from Soft Faults

- This next series of experiments focuses on showing how the effects of the fault can be recovered from
  - Goal: Establish that numerical soft fault models can not only provide comparable effects on the convergence of the algorithm, but also force the fault tolerant variant of the algorithm to respond similarly to direct injection of bit flips
- Faults are injected near the middle of program execution, at $1.4s$
- The checkpointing scheme described earlier is used with $\gamma = 1.01, 1.05, 1.25$

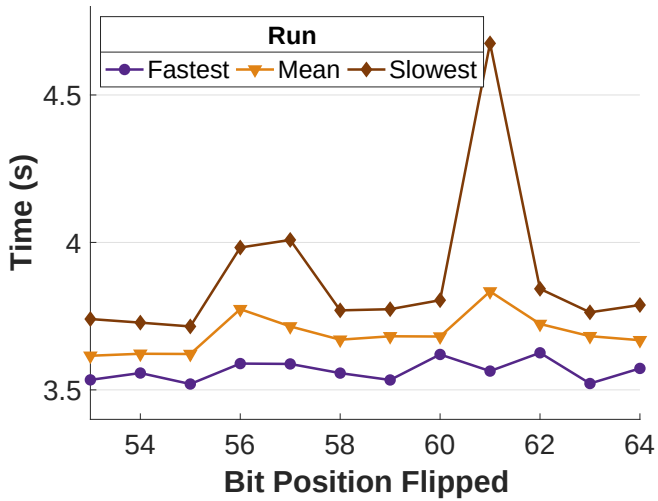
Bit Flips (exponent and sign bits only)



(a) $\gamma = 1.01$

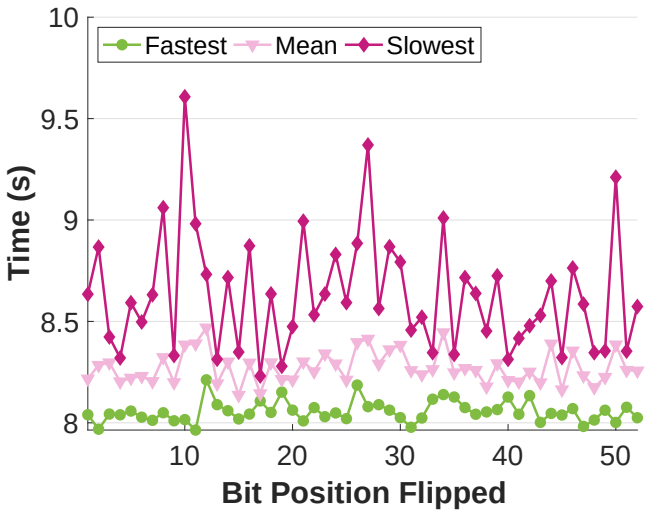


(b) $\gamma = 1.05$

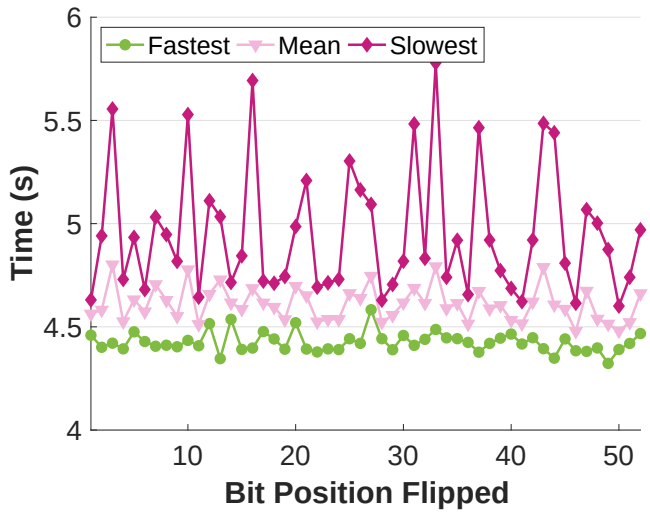


(c) $\gamma = 1.25$

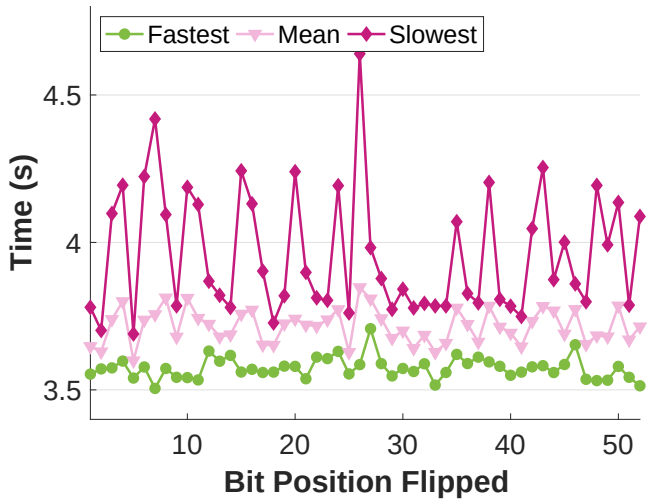
Bit Flips (mantissa)



(a) $\gamma = 1.01$

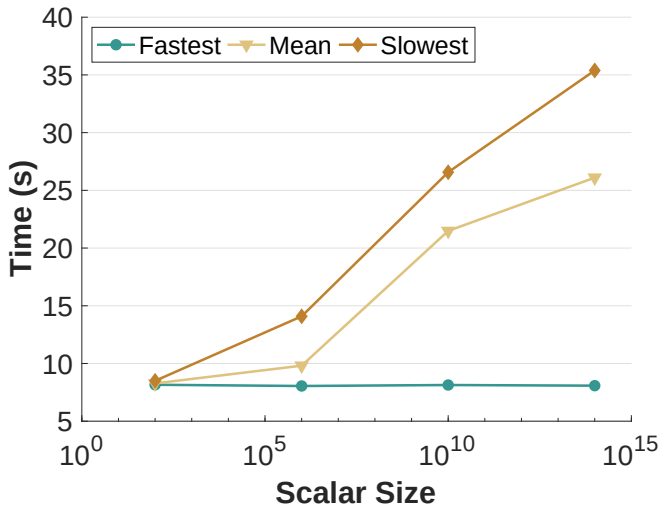


(b) $\gamma = 1.05$

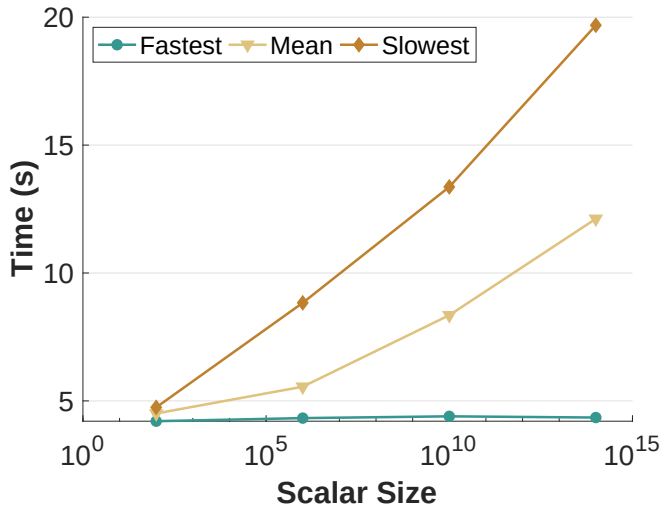


(c) $\gamma = 1.25$

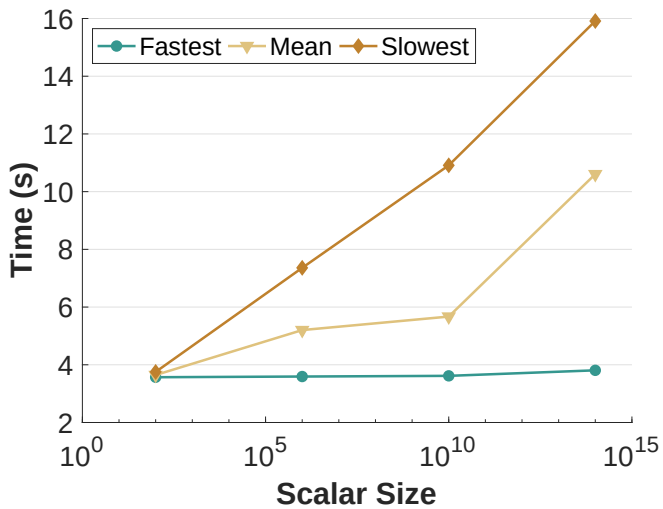
SBSFM



(a) $\gamma = 1.01$

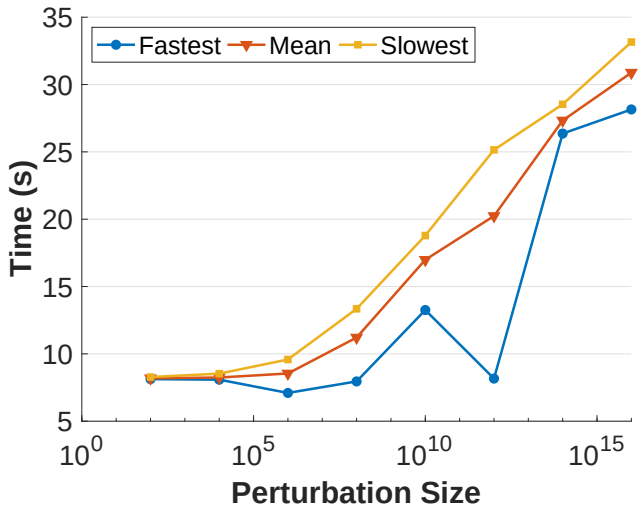


(b) $\gamma = 1.05$

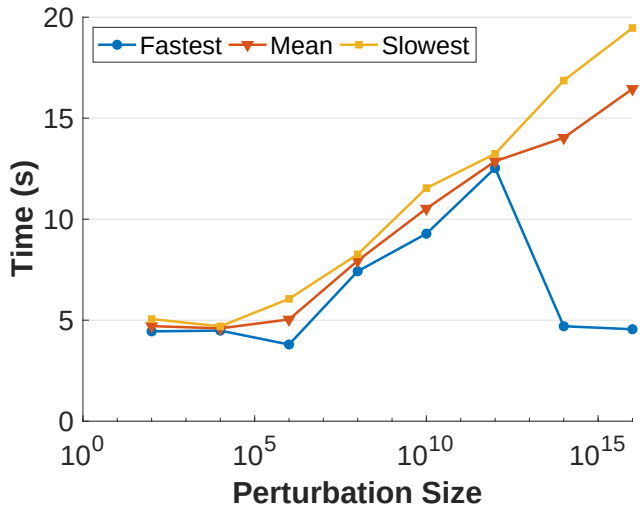


(c) $\gamma = 1.25$

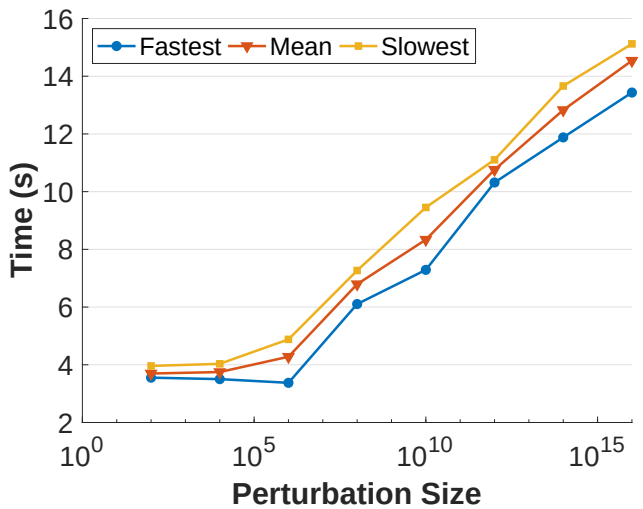
PBSFM



(a) $\gamma = 1.01$



(b) $\gamma = 1.05$



(c) $\gamma = 1.25$

Conclusions

- Both numerical soft fault models can be tuned to have similar impacts (compared with bit flips) on hybrid parallel asynchronous iterative methods
  - Both on the classical algorithm and on the fault tolerant variant
- The parameters in the numerical soft fault models allow the size of their impact to be tuned by a user
- The large difference between impacts caused by bit flips in exponent and sign bits compared with mantissa bits suggests that the average effect of flipping a bit may not be sufficiently bad to force an algorithm to respond

Future Directions

- Future directions:
  - Perform the same series of experiments on a wide variety of problems
  - Test other asynchronous solvers
  - Experiment with different recovery strategies
  - Explore the propagation of the effects of the faults amongst the components
    - Specifically: how does the rate at which subdomains interchange information relate to the impact and detection of a fault as well as recovery from the effects of a fault?

Thank you

Questions?