Uncertainty Quantification in High Performance Computing

Florian Augustin and Youssef M. Marzouk

Massachusetts Institute of Technology
Department of Aeronautics and Astronautics

PLDI – APPROX Workshop 2014
Edinburgh, United Kingdom
June 13, 2014

Motivation and Objective

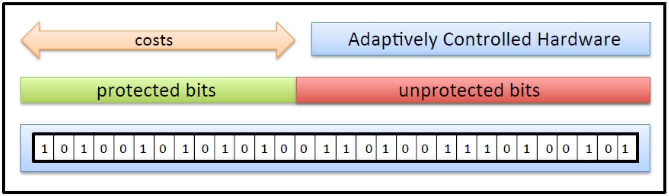
- Faults are inevitable in high-performance computing
- Fault handling is an active and diverse field of research:
  - **Reliability:**
    - Quantify the rate of faults
    - Allocate reliable and unreliable memory/operations to meet a user-specified level of reliability
    - Use checkpointing, redundancy, checkers, etc. to detect faults
  - **Accuracy:**
    - Exploit algorithm-dependent properties (conditioning, contraction, etc.) to allow for *small* faults [Stoyanov and Webster 2013]
    - Reformulate existing algorithms to make them *fault-resilient* [Bridges et al. 2012]
    - Allow users to specify an *acceptable level of variability* in the results of a code executed on unreliable hardware

Can uncertainty quantification help **trade acceptable inaccuracies for efficiency?**

Accuracy Scaling: Adaptively Controlled Hardware

A *possible way* of **controlling accuracy**:

- Hardware design may expose **tuning controllers** ν to the software engineer

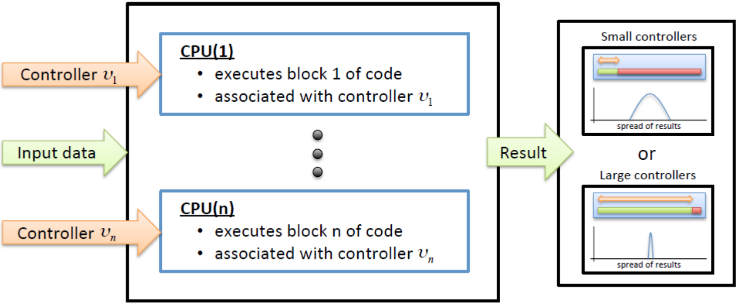


- Allow for **scaling of fault magnitudes** by setting each controller $\nu \in [0, 1]$
  - $\nu = 0$ means that no controller is applied; the code is executed on completely unreliable hardware
  - $\nu = 1$ means that all faults are filtered; the code is executed on completely reliable hardware
- Associate a **cost function** with the controller setting, for instance determined by the hardware specification

Accuracy Scaling: Adaptively Controlled Hardware

A *possible way* of **controlling accuracy**:

- Partition user code into *n* blocks
- **Execute each block on a separate unit** that is assigned to a controller ν_i, i.e., with a user-prescribed allowable fault magnitude



- Spread of possible results of the code can be controlled by adjusting the controllers $\{\nu_1, ..., \nu_n\}$

Accuracy Scaling: Adaptively Controlled Hardware

Example: Singular values of a matrix A after a fault-corrupted QR decomposition

- Subdivide QR decomposition algorithm into 3 blocks
  - Compute Householder matrices ▶
  - Compute $R = Q_n \cdots Q_1 A$
  - Assemble $Q = Q_1^T \cdots Q_n^T$

```
for (int i = 0; i < n-1; i++) {
    // block 1 : computing local matrix Q_k
    vec_norm = 0e0;
    for (int j = i; j < n; j++) {
        u[j] = R[j*n + i];
        vec_norm = vec_norm + u[j]*u[j] +
            noise(controllers[0], noise_rate);
    }
    vec_norm = sqrt(vec_norm);
    if (u[i] > 0e0) vec_norm = -vec_norm *
        (1e0 - noise(controllers[0], noise_rate));
    u[i] = u[i] - vec_norm;
    vec_norm_s = 0e0;
    for (int j = i; j < n; j++)
        vec_norm_s = vec_norm_s + u[j]*u[j] *
        (1e0 - noise(controllers[0], noise_rate));
    for (int j1 = i; j1 < n; j1++) {
        for (int j2 = i; j2 < n; j2++) {
            Qtmp[j1*n + j2] = -2e0*u[j1]*u[j2]/vec_norm_s +
                noise(controllers[0], noise_rate);
            if (j1 == j2) Qtmp[j1*n + j2] = Qtmp[j1*n + j2] +
                1e0 * (1e0 - noise(controllers[0], noise_rate));
        }
    }
```

- **Fault model:**
  - Frequency of faults: 1 per 10^5
  - Magnitude of faults: $1.0 - \nu_i$

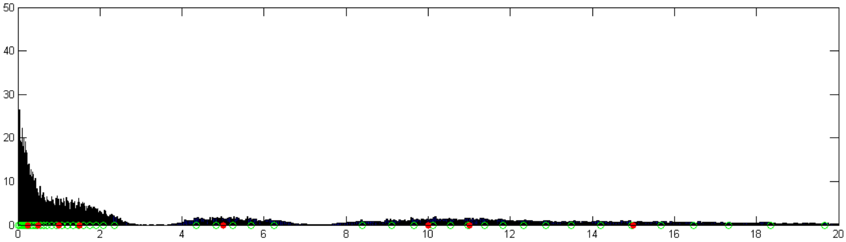
- **Application:** $A \in \mathbb{R}^{50 \times 50}$ with 8 distinct singular values
  - How much fault corruption can enter the computation while allowing correct reconstruction of the singular value multiplicities?

Accuracy Scaling: Adaptively Controlled Hardware

Example: Singular values of a matrix A after a fault-corrupted QR decomposition

Perform 1000 runs with a fixed spectrum and a random eigenbasis for each sample

- Controllers $\nu = \{0.0, 0.0, 0.0\}$, i.e., the code is executed on unreliable hardware



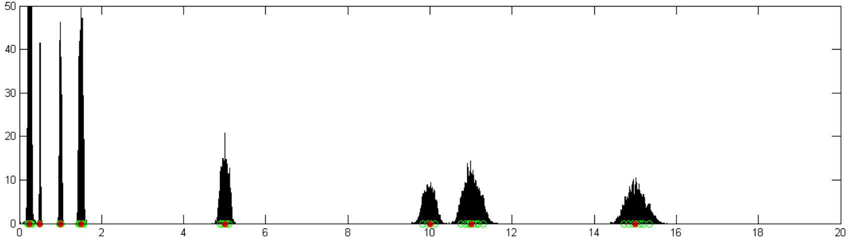
- No reconstruction of the rank is possible
- Matrix may become singular because of faults
- Conclusion: QR decomposition should be computed with greater accuracy

Accuracy Scaling: Adaptively Controlled Hardware

Example: Singular values of a matrix A after a fault-corrupted QR decomposition

Perform 1000 runs with a fixed spectrum and a random eigenbasis for each sample

- Controllers $\nu = \{0.9, 0.9, 0.9\}$,
  i.e., the code is executed on nearly-reliable hardware



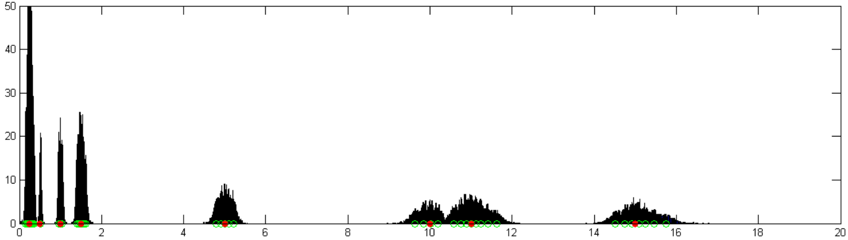
- Reconstruction of rank/multiplicities is possible
- Can the QR decomposition can be computed more efficiently (smaller ν)?

Accuracy Scaling: Adaptively Controlled Hardware

Example: Singular values of a matrix A after a fault-corrupted QR decomposition

Perform 1000 runs with a fixed spectrum and a random eigenbasis for each sample

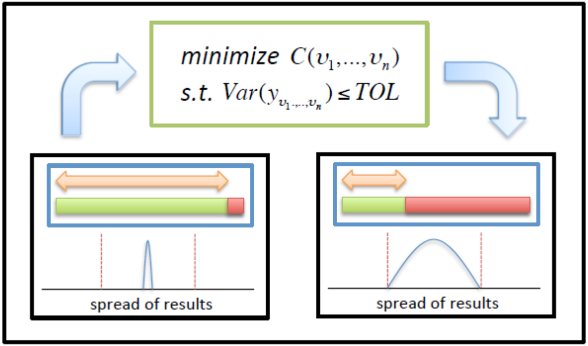
- Controllers $\nu = \{0.8, 0.8, 0.8\}$,
  i.e., the reliability of the hardware is adjusted to the code



- Reconstruction of rank/multiplicities is possible
- Increased efficiency while maintaining sufficient accuracy

Finding the Optimal Controllers

- User specifies **acceptable variability** TOL of results/outcomes y
- Find controllers $\{\nu_1, \ldots, \nu_n\}$ that **minimize the costs** C and **meet the tolerance TOL on variability** of the results $y_{\nu_1,\ldots,\nu_n}$



- **Offline:** compute and store controllers in a library (e.g., using DFO)
- **Online:** pick suitable controller for the application at hand

Example: Prothero-Robinson ODE with Contractive Paths

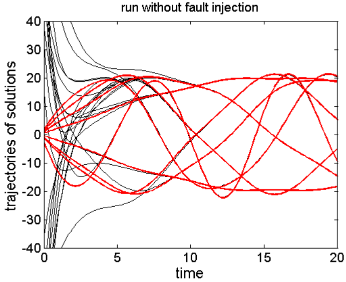
- Linear system of differential equations with system matrix A
- Attraction of trajectories to stable paths if A only has negative eigenvalues
- Application in modeling chemical and biological processes

- Find controllers ν_1, ν_2, ν_3 such that **average variance** of the eigenvalues of A is **less than 1%**

Optimal controllers:

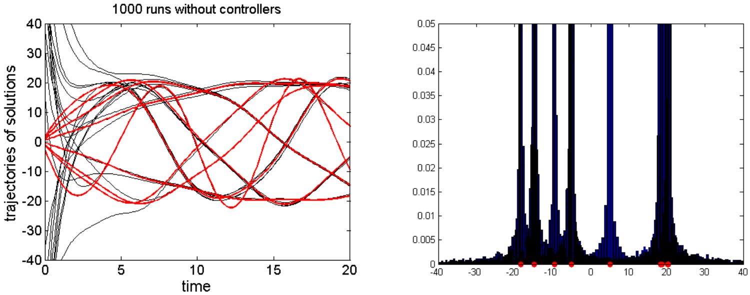
ν_1	ν_2	ν_3
0.7237	0.5657	0.7608

- **Exact solution** (without fault injection)
    - 8 stable paths (red lines)
    - 20 solution trajectories (black lines)
    - Solver: implicit Euler method



Example: Prothero-Robinson ODE with Contractive Paths

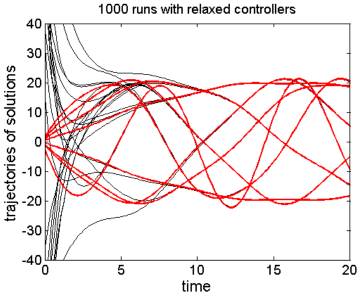
- **Average trajectories** over 1000 runs on ν-reliable hardware
- $(0, 0, 0)$-reliable hardware:

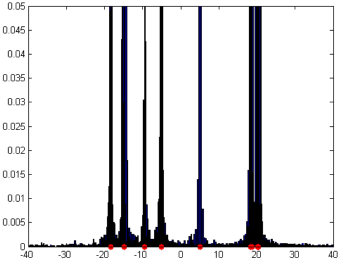


  - Averages of fault-corrupted trajectories *(left)*
  - Distribution of solutions at final time $t = 20$ *(right)*; red dots indicate the exact solution
  - Failed solutions (exceeding 10^3) are filtered out (9.78% of runs)

Example: Prothero-Robinson ODE with Contractive Paths

- **Average trajectories** over 1000 runs on ν-reliable hardware
- $(0.7237, 0.5657, 0.7608)$-reliable hardware:

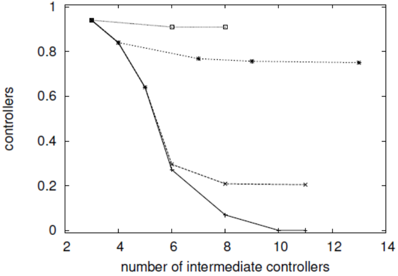




  - Averages of fault-corrupted trajectories *(left)*
  - Distribution of solutions at final time $t = 20$ *(right)*; red dots indicate the exact solution
  - Failed solutions (exceeding 10^3) are filtered out (3.82% of runs)

Example: Jacobi Iterative Solver for the Helmholtz Equation

- **Solution of the discretized Helmholtz equation**, $\Delta u - \alpha u = f$, on $[-1,1]^2$ with Dirichlet boundary condition $u\big|_{\partial[-1,1]^2} = 0$
- **Jacobi iterative method**, i.e., solve $Au = F$ using the iteration $Du_{k+1} = F - Ru_k$, $A = D + R$
- Use only one controller ν and apply 100 Jacobi iterations
- Optimize controller such that $\mathbb{V}\text{ar}[\|u_{100} - u_{99}\|^2] \leq \tau$ for $\tau \in \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$ (bottom to top in figure)
- Tighter tolerances yield more costly controllers



UQ for HPC: Opportunities and Limitations

- Allow for controlled inaccuracies to **increase efficiency**, using a probabilistic approach
- Potentially **expensive offline phase** to create a library of controllers
- **Cheap online phase** while executing the code
  - **Soft errors** have reduced impact on results due to controllers
  - Potential for **faster performance** due to fewer hard failures
- Approaches to achieve accuracy thresholds may not be general purpose tools, but rather seem **problem-specific**
  - Should then exploit common **numerical properties** like conditioning, damping, contraction, etc., to generalize the applicability of controllers
  - Possibly provide **toolbox for frequently occurring sub-problems** (e.g., iterative solvers for large systems, dissipative differential equations)
- **Open questions**:
  - Do controllers yield similar results for a broad class of problems?
  - Will future hardware allow for tunable accuracy?
  - What kind of fault models are reasonable?
  - Will a notion of 'controller cost' be available?