Hardware and Software Considerations for VV&UQ

David E. Bernholdt
Oak Ridge National Laboratory
bernholdtde@ornl.gov
What to Expect in this Talk

• A look at fault tolerance, now and in the future
  – More reasons why V&V and UQ will become increasingly important

• A look at some software techniques that may help
  – Developing an infrastructure that makes V&V and UQ easier
  – Looking primarily inside the applications

• Don’t expect solutions
  – There is much research to be done in this area
Not All Faults Lead to Fail-Stop Situations

• It is common in HPC to think about fault tolerance in terms of fail-stop situations
  – A node overheats and shuts down
  – A disk refuses to spin up
  – A network router dies

• Checkpoint/restart is the most widely used FT technique in HPC
  – Primarily to mitigate fail-stop

• But what about faults that progress to errors, but not to failures?
  – Especially silent data corruption!
Errors Abound!

- **CPU**
  - Disabled ECC on ASCI Red processors due to firmware bug, manifested in numerical error in Linpack benchmark execution (Constantinescu, 2000)

- **Memory**
  - IBM Blue Gene/L had problems with parity errors in L1 cache. ~8 hr MTBF for 64k node job, ~5 hrs for 104k nodes (Glosli, 2007)

- **Storage**
  - 3.45% of 1.53 M drives developed latent sector errors over a 32-month period (Bairavasundaram et al., 2007)
  - Read, write, compare 2 GB file every 2 hrs for 5 weeks on 3000 nodes found 500 errors on 100 nodes (Panzer-Steindel, 2007)

- **Network**
  - TCP/IP checksums and link-level CRCs miss errors in between one packet in $16 \times 10^6$ and one in $10 \times 10^9$ (Stone & Partridge, 2000)
  - *Note:* on 10GigE: $16 \times 10^6$ packets in 2 min, $10 \times 10^9$ in < 24 hrs
The Trends are in the Wrong Direction

• Origins of soft errors in logic devices
  – Cosmic rays, radiation, power fluctuations, temperature fluctuations, electrostatic discharge, manufacturing variances, aging and breakdown, …

• Trends in VLSI components
  – Number of dopant atoms per transistor dropping exponentially
  – Variability increases
  – Soft-error rates increasing 8% per year

Figures from Borkar, 2005

• Aggressive voltage scaling for power reduction
  – More susceptible to fluctuations
Hardware-Only Solutions May Not Be Feasible

• Most error detection and recover schemes have high performance, power, and hardware costs (sometime > 200%) for 100% tolerance

• Simpler checkers can achieve 90% coverage with 15-20% hardware cost, with power separately controllable (Pan et al., 2008)
Connection to V&V and UQ

- V&V and UQ may take place in “hostile” conditions
  - Distinguish hardware errors from software errors
  - Distinguish errors from “natural” variation

- Careful characterization of code permits better detection of errors when they occur

- Need new ways of thinking about software design to account for V&V and UQ and error detection/correction
  - Detect silent data corruption
  - Containment of errors (fail early)
  - Make testing and comparisons easier
  - Generally defensive programming
Software Techniques to Consider

To address SDC problems, generally support V&V and UQ
  – Within applications rather than “outside”

• Fault tolerance-driven approaches
  – Redundant execution
  – Algorithm-based fault tolerance

• Interface contracts

• Component-based software development
Redundant Execution

- 2x redundancy allows error detection, 3x+ correction
- Some possible implementation strategies
  - Operating system level, e.g. PLR *(probably not MPI-compatible)*
  - MPI-level, e.g. rMPI, P2P-MPI, Volpex MPI
  - Application-level, *not known (to DEB) in the wild*
  - User-level, e.g. *run it twice!*
- Status: not production, but could be soon
  - I/O and related externalities
  - Checking for discrepancies
- Pros
  - Should catch all SDC
- Cons
  - Expensive
  - Hard to use selectively
  - Hard to get it right (esp. communication)

PLR with 3 processes, from Shye, et al., 2009

rMPI mirror (l) and parallel (r) protocols, from Brightwell, et al., 2010
Algorithm-Based Fault Tolerance (ABFT)

- Build capability to detect, correct errors into solution algorithms

- Some possible implementations
  - Natural fault tolerance
  - Reduced models as “backup” processes
  - Build checksums into computations
ABFT through Natural Fault Tolerance

- Choose (or design) a solution algorithm with mathematical properties to naturally withstand faults of concern
- Status: Demonstrated for selected algorithms for fail-stop node faults (Engelmann & Geist)
  - Mesh-free chaotic relaxation (Laplace/Poisson), finite difference/element methods, dynamic adaptive refinement, asynchronous multi-grid methods, Monte Carlo method, peer-to-peer diskless checkpointing, global peer-to-peer broadcasts of values, global maximum/optimum search, Locally Self-consistent Multiple Scattering (LSMS) method, Molecular dynamics
- Not yet studied for data corruption errors
  - Would depend on different mathematical properties

From Engelmann, 2005
ABFT through Reduced Backup Models

• Use simplified or reduced fidelity version of model as a backup process
• Primarily used in real-time environments (e.g. target tracking)
  – Primary process fails or misses deadline
  – Backup runs quickly, “just good enough” to bridge to next step
• Status: just thought of it (in this context)
  – Can reduced model track full model well enough for error detection while staying cheap?
• Pros
  – Similar to replicated execution, but cheaper
• Cons
  – Requires that you know your model very well
  – Effectiveness for SDC unknown
ABFT through Checksums

- Augment simulation data with checksums that can be carried through operations
  - Detect errors, recover from errors or lost data
- Status: demonstrated for many linear algebra algorithms (Dongarra et al.)
  - AXPY, SCAL (BLAS1); GEMV (BLAS2); GEMM (BLAS3)
  - LU, QR, Cholesky (LAPACK); FFT
- Pros
  - Relatively inexpensive
  - Overhead scales with number of faults tolerated
- Cons
  - Needs to be extended beyond linear algebra
  - May not catch all SDC (coding issues)

From Bouteiller, 2009
Interface Contracts

- Routine carries with it a set of pre- and post-conditions and invariants as part of interface
  - Prerequisites for correct usage, guarantees for results

```cpp
double dot (in array<double> u, in array<double> v, 
in double tol)
throws sidl.PreViolation, sidl.PostViolation;

require /* Preconditions */
  u_is_1d: (u != null) implies (dimen(u) == 1;)
  v_is_1d: (v != null) implies (dimen(v) == 1;)
  same_size: size(u) == size(v); 
  non_negative_tolerance: tol >= 0.0;

ensure /* Postconditions */
  areEqual(u, v, tol) implies (result >= 0.0);
  (isZero(u, tol) and isZero(v, tol))
    implies nearEqual(result, 0.0, tol);
```

Method parameters and returns

“Built-in” methods (user-provided methods also possible)
Connection to V&V and UQ

• Could be used to capture V&V and UQ information
  – Preconditions to ensure usage is consistent with V&V’ed range
  – Postconditions for properties of output expected mathematically, and through V&V and UQ
  – Violation of postconditions may indicate data corruption

• Possible implementation strategies
  – Macros (relatively common)
  – Libraries (+ helpers) (CCA/Babel, a few others)
  – Language built-ins (but not in traditional HPC languages)

• Pros
  – Other uses besides SDC, VVUQ
  – “Executable documentation”

• Cons
  – Hard to distinguish small magnitude SDC (coverage not 100%)
  – Contract clauses may be expensive
Contract Enforcement is a Concern

- Enforcement of contracts is overhead on application

- Common practice is to enforce contracts during development, but not during production
  - Trade correctness for performance
  - Implicitly assumes development & testing provide same conditions as production

- Contracts cannot catch violations if they are not enforced!
  - Production often stresses codes more than development & testing
  - No chance to catch SDC unless contracts are enforced
Selective Enforcement in Babel  
(Tammy Dahlgren, LLNL)

Provide enforcement policies besides “always” and “never”

• Note that clauses vary in computational complexity
  – Constant time, simple expressions, linear time, method calls (unknown cost), …

• Enforcement by complexity classification
  – Whatever you think you can afford or will be most useful

• Sampling of clauses
  – Random, periodic, …

• Performance-constrained enforcement
  – Set a limit on how much to spend on enforcement (i.e. 10% time)
  – Sample clauses to maximize coverage within limit (different schemes)

• Not 100% coverage, but better than nothing

14 November 2010
**Example: Performance-Driven “Adaptive Timing” Sampling Policy**

Test programs:
- MT: volume mesh (ITAPS)
- A, MA, AA: simplicial mesh (ITAPS)
- VT: vector (Babel)
- Adaptive Timing policy targets 10% overhead, biases enforcement towards “fast” clauses

*From Dahlgren, 2007-2008*
Open Questions for Contracts

• What additional contract capabilities would be useful for V&V and UQ?

• How to most effectively express and test VVUQ-related contract clauses?

• Selective enforcement strategies to increase detection of SDC?
Component-Based Software Development (CBSD)

- Latest in a series of programming concepts to help software developers and architects deal with complexity
- Components are units of software that...
  - Encapsulate specific functionality
  - Expose that functionality via well-defined interfaces
- Internals of components are opaque to other components
- Provides a “plug and play” concept for software
  - Components providing the same interface are interchangeable
Connection to V&V and UQ

• Design your software so that you can swap out the implementations of key parts for other implementations when you need to

• “Toolkit” of interchangeable components for different types of problems

• Replace a simulation component with one that replays experimental results

• Cross-code comparison
  – Assessed reference code
  – Alternative algorithms

• Side by side comparisons (selective redundant execution)
Computational Facility for Reacting Flow Science (CFRFS)

- A toolkit to perform simulations of lab-sized unsteady flames
  - Solve the Navier-Stokes w/detailed chemistry
  - Various mechanisms up to ~50 species, 300 reactions
- More than 100 components in toolkit
- ~40 needed for typical simulations

CCA-based combustion application "wiring diagram" and results. Courtesy Cosmin Safta, (SNL)
Center for the Simulation of RF Interactions with Magnetohydrodynamics (SWIM)

- Integrated modeling of RF, heating, transport, and magnetohydrodynamics in fusion plasmas
- Multiple components for each class of physics
  - e.g. AORSA and TORIC
- “Replay” components
Open Issues for Component-Based Software Development

- CBSD is more a development approach than a set of tools
  - Concepts influencing software architecture
  - Benefits for VVUQ are indirect

- True plan-and-play of components is challenging
  - Different algorithms often require different inputs

- V&V and UQ for components vs whole applications (or specific problems)
Summary

• Increasing likelihood of silent data corruption on future hardware may become problematic
  – Unlikely to be dealt with entirely in hardware

• V&V and UQ of applications likely to take place in a “hostile” environment

• Software mechanisms are available to help catch silent data corruption
  – Redundant execution (100% coverage possible)
  – Algorithm-based fault tolerance
  – Contracts
  – Component-based software development

• Need to think about software in new ways to take advantage of them