High Performance Computing Division

Inexact Computing for Pre-Post-Moore's and Post-Moore's Time Frames

Laura Monroe
High Performance Computing Design Group
Los Alamos National Laboratory

LA-UR-17-23438

Introduction/Background

- Due to physical limitations of feature size, inexact computing might be necessary and/or advantageous for pre- and post-exascale supercomputing
  - Fragility to faults
  - Power needs
  - Performance goals
  - New opportunities
- We believe that inexact computing may provide new approaches to old problems and provide solutions to previously unsolved problems
- We are developing non-deterministic methods and support in service of LANL computational and mission needs
  - Pre-exascale: power and performance enhancement
  - Post-exascale and post-CMOS: the above, plus novel computing

L. Monroe, J. Daly, N. DeBardeleben, S. Michalak, Q. Guan, and K. Rudd. "Probabilistic Computing for HPC in the Post-Moore's Era". Post-Moore's Era Supercomputing Workshop. PMES 2016, November 2016.

L. Monroe, "Probabilistic Computing", IEEE Computer, December 2015.

Talk: L. Monroe, "Probabilistic and Approximate Computing", IEEE Rebooting Computing Summit 4, Washington DC, 2015.

Dealing with Non-Determinism

- Exact computation
  - Do you demand a correct answer, determinism throughout, and the same computation every time? *EDAC*.
- Inexact computation
  - Or do you permit probabilistic transitions, and different computational paths each time? *Probabilistic computing*.
  - Or do you terminate your calculation, coming “close” each time (but deterministically)? *Approximate computing*.
- Or do you consider all three approaches?
  - *This is a good path.*
  - Allows all the benefits of EDAC and the freedom of inexact approaches
- The difference is the computational paths taken
  - Probabilistic has to do with the transitions and the branching
  - NOT the state nor the data

Probabilistic Turing Machine

- Turing machine
  - The transitions are fully determined by state and input
- Non-deterministic Turing machine
  - The transitions the machine makes are not fully determined by the state and input.
  - In other words, there is more than one possibility at some of the transition points.
- Probabilistic Turing machine
  - It is an NTM and the transitions are determined by state, input, and a probability distribution function.
- The theory is important because it is the basis for the language we are using, and is the basis for the computational complexity theory in use

Introduction/Background: Types of Inexact Computing

Probabilistic	**Approximate**
Non-deterministic, probabilistic transitions: different answer each time	Deterministic transitions: same answer each time
Full resolution	Reduced resolution/convergence
Accurate	Precise
Could give power savings, better resilience	Could give power savings, better resilience, has potential for faster convergence

Introduction/Background: Conceptual Challenges and Our Approaches

- Inexact computing works well with signal processing and other less demanding problems, where the inexactness can be hidden in the noise
    - Can we apply these techniques to ASC physics problems and integer problems of interest? This is pushing the envelope.
- Probabilistic computing fundamentally breaks the social contract between hardware engineers and software engineers
    - We are developing design patterns for inexact computing to mitigate the concern/confusion around inexactness
- The mathematical bridge between device physics and computing is not being developed globally. The bridge needs attention.
    - The mathematical foundations of computing are currently not taught to hardware designers
    - Faultiness of the hardware is not being taught to the software engineers
    - We are addressing both of these, but more needs to be done in the post-CMOS time frame.

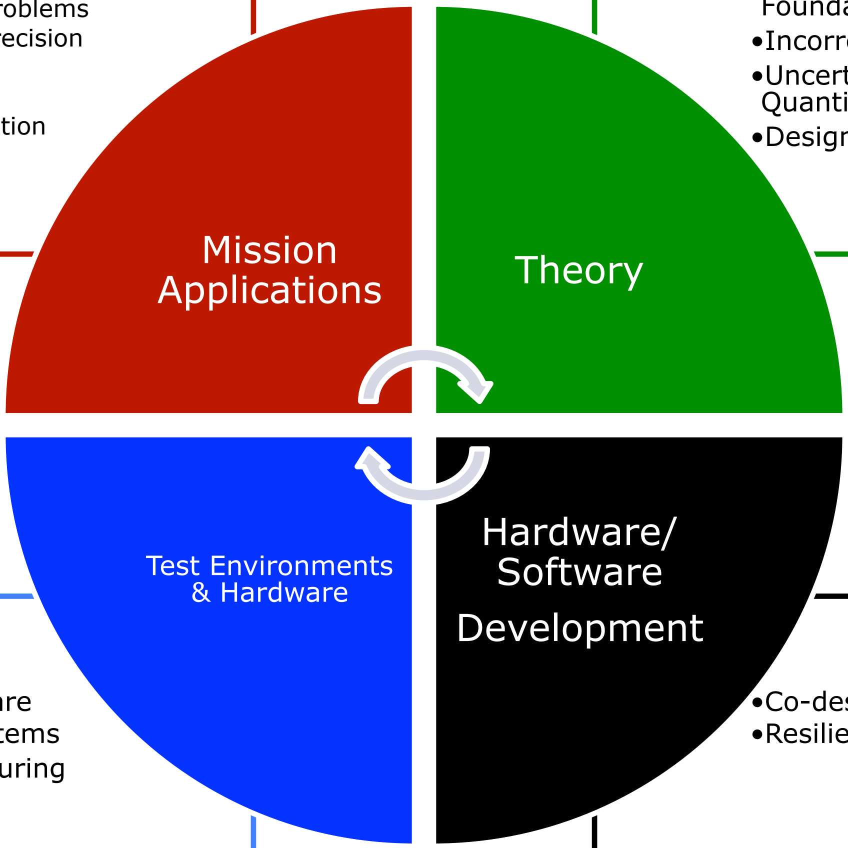
Techniques

Motivating Examples

- LDRD Sort/search
- Integer Math Problems
- ASC Variable Precision
- ASC Stochastic Simulations
- ASC Approximation

- Mathematical Foundations
- Incorrectness Models
- Uncertainty Quantification
- Design Patterns

Current Work



Future Work

- Faulty Hardware
- Designed Systems
- Probabilistic Turing Machine

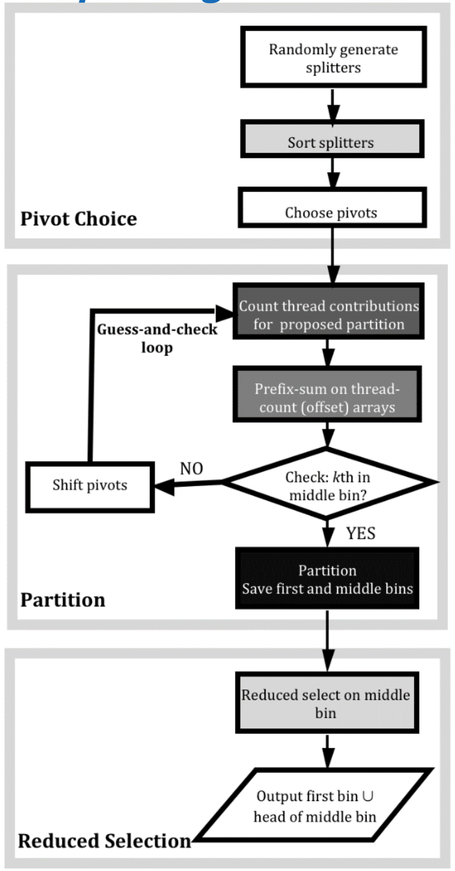
- Co-design
- Resilience Methods

Top K Search

- Top K is a search algorithm used to find the largest k elements in a list
  - Often implemented as a sort
  - Algorithm was redesigned as a parallel probabilistic selection algorithm (Lazy Select)
- Probabilism provides
  - An embarrassingly parallel implementation
  - The ability to load balance on a many-core architecture
  - A perfect match to the Fermi architecture: **3X speedup** from the best selection previously known on GPU, up to **64M threads/kernel**, and processed **4x the previous best list size** in GPU memory
- This algorithm uses some common probabilistic design patterns
  - Probabilistic input conditioning on pivot choice
  - Looping for executing multiple iterations
  - Control flow includes a correctness "check"
  - "Do-over" design pattern

L. Monroe, J. Wendelberger, S. Michalak. "Randomized Selection on the GPU"HPG 2011, August 2011.
C. Lunardi, H. Quinn, L. Monroe, D.Oliveira, P. Navaux, P. Rech. "On the Error Criticality of Sorting Algorithms for HPC and Large Servers Applications", Radiation Effects on Components and Systems, RADECS 2016, August 2016.

Top K Algorithm


Randomly generate splitters
Sort splitters
Choose pivots
Pivot Choice
Count thread contributions for proposed partition
Guess-and-check loop
Prefix-sum on thread-count (offset) arrays
Check: kth in middle bin?
NO
Shift pivots
YES
Partition Save first and middle bins
Partition
Reduced select on middle bin
Output first bin ∪ head of middle bin
Reduced Selection

Integer Math

- Our current goal is to assess probabilistic integer math for possible improvements in power efficiency in current or future architectures
  - Can we relax the requirements for exactness and determinism in integer algorithms?
  - Could we design math units that are higher performance and more tolerant to faults than traditional integer math?
- Focus on simple integer problems to investigate different probabilistic and resilience algorithms
- Using both faulty hardware and probabilistic computer to experiment and prove the workability

ASC Variable Approximate Computing

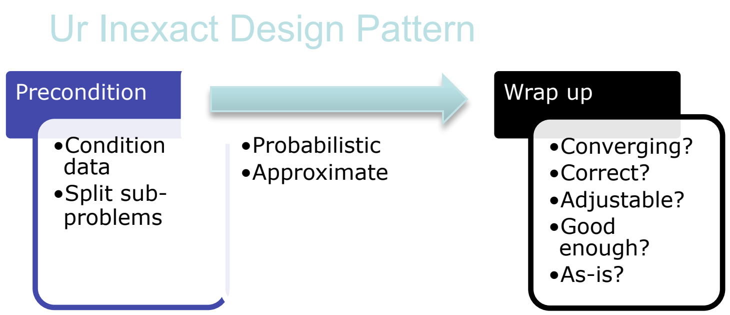
- ASC's floating point applications could also benefit from inexact math
  - Variable precision floating point circuits is common in FPGAs
  - Adapting ASC sub-problems for variable precision floating point saves power/time/space, as long as the loss of precision is reasonable
- Current work
  - Have identified a mini-app instrumented for precision (CLAMR)
  - Need to characterize mini-app over a wide range of precision, including full precision
  - Analyze the tradespace for precision, correctness, performance, and power

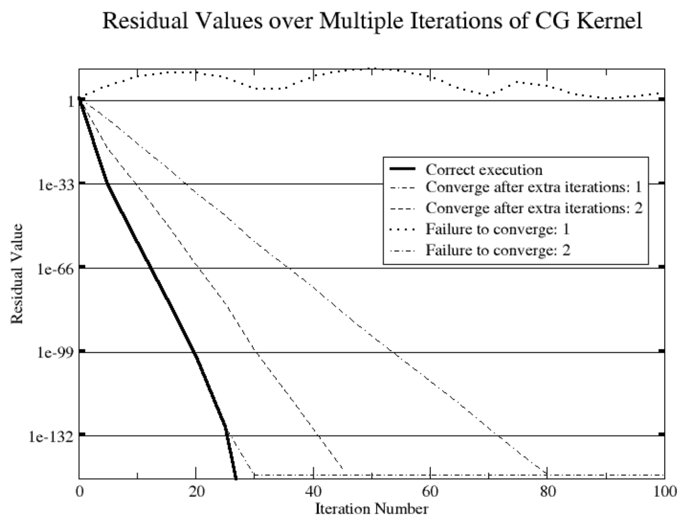
Theory and HW/SW Development

- Inexact computing needs
  - Strong mathematical foundation
  - In-depth collaboration between hardware/software development and the fault model
  - Design patterns to help translate deterministic algorithms into inexact algorithms
- Mathematical foundations and incorrectness models can provide understanding of how the inexactness affects the calculation
  - Weed out poorly performing inexactness quickly
  - Develop mathematically sound resilience methods for “checking” incorrectness

Reference Design Patterns

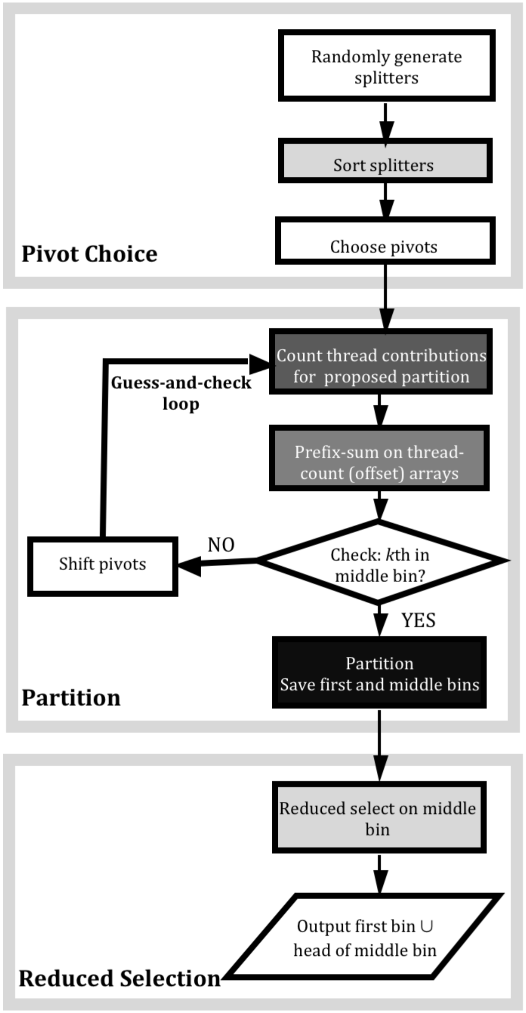
- Abstracted from mission applications to identify design patterns for inexact computing
  - Can see the basic structures that work well in probabilistic/inexact
  - Can see the guiding abstractions to make algorithm design tractable
  - Can see how to design domain problems
- We are developing a set of design patterns for inexact computing (hardware/software/both)
  - Our current set is abstracted from mission apps
  - We believe that these design patterns will be foundational for future development
- We have found that many inexact design patterns have a looping structure, and many incorporate feedback
  - Loop control keeps the main looping structure iterating until the answer is correct, reasonable, good enough or just taken as is
  - In previous work, we found that these iterative structures are often resilient to faults: they often converge with extra time

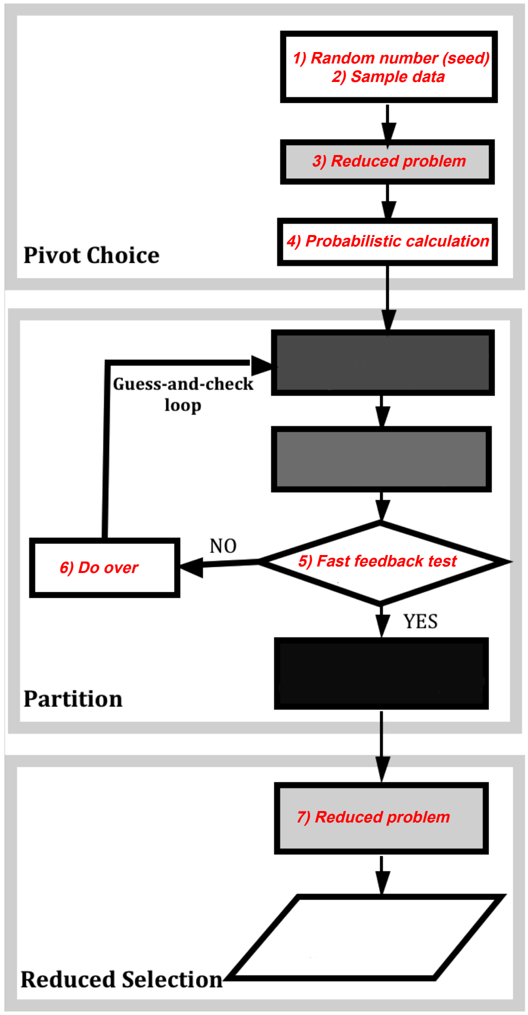




H. Quinn, T. Fairbanks, J. L. Tripp and A. Manuzzato, "The Reliability of Software Algorithms and Software-Based Mitigation Techniques in Digital Signal Processors," *2013 IEEE Radiation Effects Data Workshop (REDW)*, San Francisco, CA, 2013, pp. 1-8

Design Patterns: Top K Search


Randomly generate splitters
Sort splitters
Choose pivots
Pivot Choice
Count thread contributions for proposed partition
Guess-and-check loop
Prefix-sum on thread-count (offset) arrays
Shift pivots
NO
Check: kth in middle bin?
YES
Partition Save first and middle bins
Partition
Reduced select on middle bin
Output first bin ∪ head of middle bin
Reduced Selection



Interplay Between Resilience Methods and Inexact Computing

- Many of the design patterns for inexact computing have some form of check
- The check construct is a control flow structure that
  - Controls continued execution
    - Has the answer converged? Is the answer good enough?
  - Controls the quality of the result
    - Can we accept this result? Do we flag this result as incorrect?
- This type of checking could develop new resilience methods based on not necessarily correction, but acceptance criteria
- We have developed enhanced ECC methods and are working toward better checks on various computations

Vilas Sridharan, Nathan DeBardeleben, Sean Blanchard, Kurt B. Ferreira, Jon Stearley, John Shalf, and Sudhanva Gurumurthi. 2015. Memory Errors in Modern Systems: The Good, The Bad, and The Ugly. In Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '15). ACM, New York, NY, USA. 2015.

Experimentation Systems (Testbeds)

- The experimentation systems provide a springboard for current and future development of inexact software or hardware
  - Special-purpose inexact FPGA circuits
  - Inexact mathematical hardware circuits
  - Inexact software codes
- The engineers are freed up to implement the new hardware and/or software algorithms without implementing the entire system
- The theoreticians are freed up to work on the fault models and mathematics for the new algorithms
- An experimentation system is the foundation for both hardware and software development
  - The hardware must be extensible so that mathematical units and co-designed circuits can be instantiated with different capabilities or different models
  - The software remains open so that engineers can adapt, experiment and characterize their algorithms

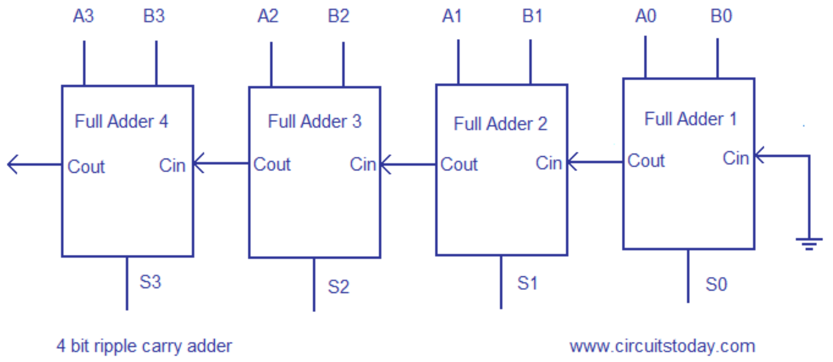
Experimentation systems under development

- Currently working on two FPGA-based frameworks for natively probabilistic computing that create probabilistic circuits out of traditional algorithms
  - Circuit corruption on an FPGA platform reusing our fault emulation environment for space-based FPGA systems: development done
  - Extensible circuits/software that allows different math libraries to be changed
- Co-designed applications would be possible through CPU interlays
  - Allow special circuits to be interfaced to the system for both hardware and software development purposes
- Looking for future possibilities for technology insertion for large-scale, in-situ demonstrations
  - Small, special-purpose clusters for HPC
  - Technology demonstrations in space or other deployed environments

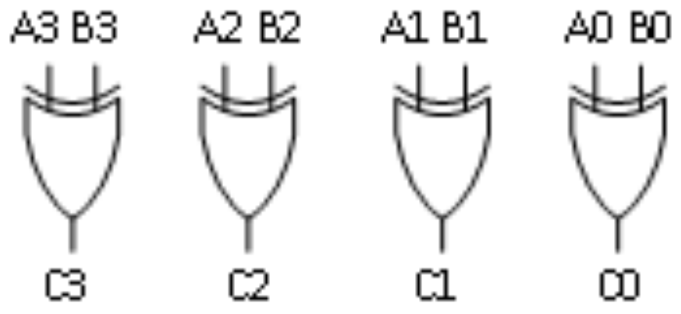
Circuit Design

- We are assessing probabilistic integer math for performance/power efficiency
  - Relax the requirements for exactne
    and determinism on integer math
  - Develop a probabilistic solution for an integer algorithm
- Currently developing different types of <u>inexact</u> math circuits
  - Hardware: higher performance inexact integer math
  - Theory: incorrectness model
  - Resilience: a method to check for or correct incorrectness
- Design patterns and the incorrectness model guide the process of designing the hardware

Ripple Carry Adder



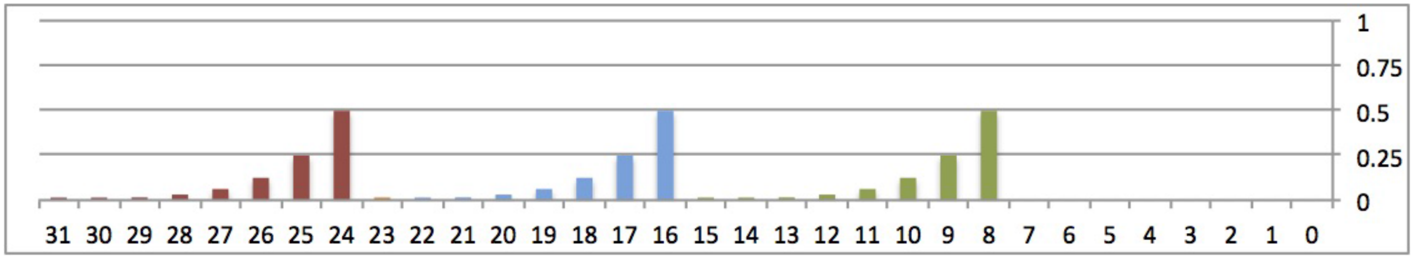
Carryless Adder



L. Monroe, W. M. Jones, C. Davis, S. Lavigne, Q. Guan, N. DeBardeleben. "On the Inherent Resilience of Integer Operators", Euro-Par Resilience 2016, August 2016

Circuit analysis

- Analyzing whether to have some number of probabilistically or algorithmically obtained carries to decrease the number of faults
  - Below is the fault model for a 32-bit add made from four 8-bit full adds without carries between the full adds



  - Each 8-bit full add has an independent fault model
  - The complete 32-bit adder has a fault model between the full adds
- The fault model is characterized by a Kronecker product, disregarding the dependence inside of a full add

$$((\frac{1}{2}-\frac{1}{2s^m})\times(1_{r-1},0))\otimes(\frac{1}{s^{m-1}},\frac{1}{s^{m-2}},\ldots,\frac{1}{s},1)$$

  - We are experimenting with the first term of the product to move from an approximate to a probabilistic adder to get closer to the exact answer

Challenges

- How much does correctness matter? How do you define correctness?
  - Application-specific **distance metrics** are necessary to address inexactness
    - Have both a "natural" application distance, and a "natural" fault distance
    - Is there interplay between them?
  - We need to have a rigorous understanding of good enough
- How does incorrectness manifest itself, specifically and statistically?
  - We need to understand fault models so that we can design algorithms for general faulty processors
- How do we design the probabilistic computing stack?
  - It is possible that engineers will adapt to incorrectness and faultiness, but it might need to be introduced slowly
  - Hardware engineers will need to be able to design them
  - Software engineers will need to be able to program them, which means design patterns, programming languages, programming models, compilers, and debuggers
  - Benchmarks to determine the tradespace between deterministic and inexact systems

All challenges at this stage, mitigation is in process….

Things have to work.

- Common agreement
  - Industry needs to sell things that work
  - Labs need to support their mission
  - Academicians need to expound things that are useful
- What does it mean "to work"? (How to attain this?)
  - Perfect correctness?
  - Faults that are (mostly) correctable?
  - Faults that are characterized and tolerable?
  - Algorithms that are close enough?
    - What does "close" mean? Need mathematically rigorous definitions
  - Faulty or probabilistic systems that still may be coded?

Economic model

- Why buy a thing that is less correct?
  - This isn’t a research question but it is basic
  - You buy it because you have to; physical reality forces it
    - Apps for which current CMOS is good enough won’t need this
  - Or because it is cheap enough to be worth the extra effort and/or the loss of correctness
- Will the average mass-market programmer be able to program probabilistic systems?
  - As opposed to special-purpose systems/apps
  - If not, will the market support these?
- What if chips become less reliable anyway?
  - “Reliability as a service”

Where to address the probabilism?

- Hardware
  - Already make it as reliable as economically possible
  - Already correct as many faults as possible
  - Control pretty much has to be correct
    - So data correctness on that processor comes with control
    - Could protect some hardware and not other
- System software
  - If OS cannot tolerate faults, need to move probabilistic to accelerators
- Application software
  - Can the application deal with the faults?

Resilience/reliability and inexact computing

- We are already there – already see errors
  - Correct in hardware as much as possible
- Processor reliability should match user needs
  - Approximate may tolerate some compromise on correctness
- Can we produce correct answers in the presence of faults?
- Must quantify unreliability before doing probabilistic

Fault models

- Essential to identify fault models: rate and distribution
- Must be done empirically and experimentally
  - Dependent on the materials, the architecture, etc
  - Specific to the hardware at hand
  - The physical world of devices not the ideal world of CS
- But then may reason about these theoretically
  - Lots of tools at hand: statistical science, algorithms etc
- A calculus of resilience would be interesting
  - Like performance complexity analysis

Programming challenges

- Coding is already complex
- Will programmers easily be able to write for probabilism?
  - An additional burden to write software to handle faults
  - How to distinguish between subroutines that accept probabilism and those that can’t?
    - i.e., sensitive to fault or insensitive
    - Systemic effects – hard to predict impact on complex system
  - They will likely take the safest, most conservative route
- Hide the probabilism?
  - Libraries
  - Error correction
  - Domain-specific languages?
  - Or quantify the probabilities
- Design Patterns
- Programming models
- Don’t forget debugging

A plea for good terminology

- Not research, but it needs to happen
- Probabilistic vs Approximate
  - Probabilistic –accuracy
    - Leverages the intrinsic probabilistic behavior of the underlying circuit fabric. or probabilistic algorithms on deterministic or non-deterministic hardware
    - Compute pathways
    - Different answer every time
    - Non-deterministic
  - Approximate – precision
    - Employs deterministic hardware designs that produce imprecise results
    - Reduced precision for example
    - Always the same answer after the same number of iterations
    - Deterministic
- Inexact computing, covers probabilistic and approximate
- Fault vs Error
  - Fault – what happens at the hardware level. Something goes wrong.
  - Error – what happens at the software or application level. The fault has an effect on the application. If it is seen, the user can re-run. If not, it is silent data corruption.

Conclusions

- Probabilistic and approximate computing may provide needed gains in resilience and power for long-term needs
  - Have developed several mission applications
    - Inexact mathematical operations
    - Stochastic simulations for effective ASC scientific computing
    - Design patterns to bridge gaps
    - A probabilistic computer for experimentation
  - Currently developing a better foundation of hardware and theory to enable faster experimentation and analysis
- This area is **cross-disciplinary** and applies to many late- and post-CMOS activities
  - The field spans theory, mathematics, algorithms, hardware and mission applications
- This is an emerging field of interest with impact for the future

Acknowledgements

- Team
  - Mathematics/algorithms: Laura Monroe (HPC-DES)
  - Computer architecture: Heather Quinn (ISR-3)
  - Architecture SME: Steve Poole (ADTSC)
  - Statistics: Sarah Michalak, Joanne Wendelberger, Lisa Moore, Jim Gattiker, Brian Weaver (CCS-6)
  - Resilience: Nathan DeBardeleben (HPC-DES), Qiang Guan (CCS-7)
  - Machine Learning: Lissa Baseman (HPC-DES)
  - Systems: Andrew Shewmaker (HPC-DES)
  - Applications: Bob Robey (XCP-2), David Daniel (CCS-7)
  - Postdocs/students: Robert Foster (CCS-6), Michael Lane, Emily Vecchia, Mitch Klein, Ryan Slechta (HPC-DES)
- Collaborators:
  - John Daly, Kevin Rudd (DOD), John Owens (UC Davis), Gary Delp, Jim Rose, Ben Buhrow, Jim Humble, Michael Lorsung (Mayo)