Uncertainty and Approximation in Computing & Data:
Raising Abstraction of Adaptation

Rajesh K. Gupta
UC San Diego.
Our love-hate relationship with non-determinism

TWO TRENDS, TWO CULTURES, TWO RESPONSES
A growing realization in Software Systems

No amount of processing power can deliver the goods to next generation of applications

• The fastest single machine with in-memory database or 100 EC2 nodes on low petabytes
  – Not enough time to be real-time
  – Not enough power to be omnipresent

• Trends
  – Stratified sampling and organization of data (BlinkDB)
  – Approximate results
    • (or even Erroneous, Palem et al)
The Problem Is

Uncertainty Means Unpredictability

• Engineered systems can not be unpredictable
• Two completely different responses
  – HW hatred and SW neglect.
• VLSI Designer: Eliminate It
  – Capture physics into models
  – Statistical or plain-old Monte Carlo
  – Manufacturing, temperature effects

• Simulate ‘degraded’ netlist with model input changes ($\Delta V_{th}$)
• Deterministic simulations capture known physical processes (e.g., aging)
• Multiple (Monte-Carlo) simulations wrapped around a nominal model.
Variability Sources: PVTA
- Static Process variation: effective transistor channel length and threshold voltage
- Dynamic variations: Temperature fluctuations, supply Voltage droops, and device Aging (NBTI, HCI)

Each one adds to the guardband
System and Software Designers: **Ignore or ask hardware to help.**

- **Architect:** Change goal to ‘averages’
  - Workload (Dynamic) Variations
  - Beginning realization that it is not the right way (e.g., Dark Silicon)

- **Software, OS:** Deny It or make it somebody else’s problem
  - Simplify, re-organize OS/tasks breaking these into parts that are precise (W.C.) and imprecise (Ave.)
  - Hardware assists (e.g., new circuit fabrics as islands: PRU in TI processors, FPGA blocks)
A realization and its consequences

• **From:** Power is deliverer of work
  • Energy, Heat, Order (Temp) $\rightarrow$ Mechanical Work

• **To:** Power is the arbiter of certainty
  – State space and computational reversibility

• Energy use provides *the* control knob to specify the level of certainty a computation can afford
  – Not just the *amount* or *speed* of computational work, but the *reliability* of the result. **Confidence.**

How do we architect our machines to expose this control knob? *(a very big question)*
Let us step back a bit: HW-SW Stack

- Application
- Hardware Abstraction Layer (HAL)
- Operating System
- Application

---
Let us step back a bit: HW-SW Stack

- Application
- Operating System
- Hardware Abstraction Layer (HAL)

Time or part
Let us step back a bit: HW-SW Stack

- Hardware Abstraction Layer (HAL)
- Operating System
- Application

20x in sleep power
50% in performance

overdesigned hardware

40% larger chip
35% more active power
60% more sleep power
What if?

Hardware Abstraction Layer (HAL)

Operating System

Application

Application

Time or part

underdesigned hardware
New Hardware-Software Interface..

- Minimal variability handling in hardware
- Operating System
- Hardware Abstraction Layer (HAL)
- Traditional Fault-tolerance
- Opportunistic Software
- Underdesigned Hardware
Variability Expeditions: UNO Computing Machines use both Modeling & Sensing

Variability manifestations:
- Faulty cache bits
- Delay variation
- Power variation

Variability signatures:
- Cache bit map
- CPU speed-power map
- Memory access time
- ALU error rates

Metadata Mechanisms: Reflection, Introspection

Do Nothing (Elastic User, Robust App)
Change Hardware Operating Point (Disabling parts of the cache, Changing V-f)
Change Algorithm Parameters (Codec Setting, Duty Cycle Ratio)
Change Algorithm Implementation (Alternate code path, Dynamic recompilation)

Models
Sensors
Navigating the space of reliability and certainty
Several Mundane but Fundamental Questions

1. How do we distinguish between codes that need to be accurate versus that can be not so?
   – How fine grain are these (or have to be)?

2. How do we communicate this information across the stack in a manner that is robust and portable?
   – And error controllable (=safe).

3. What is the model of error that should be used in designing UNO machines?
   – How do we put the error bars on our results?
Building Machines that leverage move from Crash & Recover to Sense & Adapt
Thrusted traverse institutions on testbed vehicles seeding various projects

Group A: Signature Detection and Generation
- Characterizing variability in power consumption for modern computing platforms, and implications
- Runtime support and software adaptation for variable hardware
- Probabilistic analysis of faulty hardware
- Understanding and exploiting variability in flash memory devices
- FPGA-based variability simulator

Group B: Variability Mitigation Measures
- Mitigating variability in solid-state storage devices
- Hardware solutions to better understand and exploit variability
- VarEmu emulation-based testbed for variability-aware software
- Variability-aware opportunistic system software stack
- Application robustification for stochastic processors

Group C: Opportunistic Software and Abstractions
- Effective error resilience
- Negative bias temperature instability and electromigration
- Memory-variability aware runtime systems
- Design-dependent ring oscillator and software testbed
- Executing programs under relaxed semantics
Observe and Control Variability Across Stack

By the time, we get to TLV, we are into a parallel software context:

instruct OpenMP scheduler, even create an abstraction for programmers to express irregular and unstructured parallelism (code refactoring).
How does SW see uncertainty? Especially timing related.

- Failure->Fault->Error loop?
- Timing errors?
  - ILV, SLV, PLV, TLV take the approach to characterize errors at respective levels
- But the whole point is avoid failure
  - More energy, More work?
  - Avoid failure, recover from it, or do a plan B.
- Uncertainty in software (caused by variability) is about managing parallelism.
A range of techniques, explored, being explored, some discarded already.

**ADAPTATIONS**

1. Change circuit, microarchitecture
2. Change architecture
3. Change code (compiler)
4. Change runtime
5. Change programming
1. Change LOGIC, CIRCUIT

• The most immediate manifestations of variability are in path delay and power variations.
  – Path delay variations has been addressed extensively in delay fault detection by test community.

• With Variability, it is possible to do better by focusing on the actual mechanisms
  – For instance, major source of timing variation is voltage droops, and errors matter when these end up in a state change.

Combine these two observations and you get a rich literature in recent years for handling variability induced errors: Razor, EDA, TRC, …
Reliability Architectural Archetypes:
Model, Sense, Predict, Adapt

I. Sense & Adapt
Observation using in situ monitors (Razor, EDS) with cycle-by-cycle corrections (leveraging CMOS knobs or replay)

II. Predict & Prevent
Relying on external or replica monitors → Model-based rule → derive adaptive guardband to prevent error

15-Oct-14
Abbas Rahimi/ UC San Diego
CHARACTERIZE, MODEL, PREDICT

2. CHANGE ARCHITECTURE

Bit Error Rate, Timing Error Rate, Instruction Error Rate, ….
Consider a Full Permutation of PVTA Parameters

- 10 32-bit integer, 15 single precision FP Functional Units (FUs)
- For each FU\(_i\) working with \(t_{\text{clk}}\) and a given PVTA variations, we defined Timing Error Rate (TER):

\[
\text{TER} (\text{FU}_i, t_{\text{clk}}, V, T, P, A) = \frac{\sum \text{CriticalPaths (FU}_i, t_{\text{clk}}, V, T, P, A) \times 100}{\sum \text{Paths (FU}_i)}
\]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Start Point</th>
<th>End Point</th>
<th>Step</th>
<th># of Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voltage</td>
<td>0.88V</td>
<td>1.10V</td>
<td>0.01V</td>
<td>23</td>
</tr>
<tr>
<td>Temperature</td>
<td>0°C</td>
<td>120°C</td>
<td>10°C</td>
<td>13</td>
</tr>
<tr>
<td>Process (\sigma_{\text{WID}})</td>
<td>0%</td>
<td>9.6%</td>
<td>3.2%</td>
<td>4</td>
</tr>
<tr>
<td>Aging (\Delta V_{\text{th}})</td>
<td>0mV</td>
<td>100mV</td>
<td>25mV</td>
<td>5</td>
</tr>
<tr>
<td>(t_{\text{clk}})</td>
<td>0.2ns</td>
<td>5.0ns</td>
<td>0.2ns</td>
<td>25</td>
</tr>
</tbody>
</table>
Use supervised learning (LDA) to generate a parametric model that connects PVTA variation to clock period and classes of timing error rates.

The model makes correct estimates for 97% of times. 3% misclassification is to the high-error rate class. (i.e., it is safe)
Hierarchical Sensors Observability

• The question is what mix of monitors that would be useful?

• The more sensors we provide for a FU, the better conservative guardband reduction for that FU.

• The guardband of FP adder can be reduced up to
  • 8% (P_sensor),
  • 24% (PA_sensors),
  • 28% (PAT_sensors),
  • 44% (PATV_sensors)

In-situ PVT sensors impose 1–3% area overhead [Bowman’09]
Five replica PVT sensors increase area of by 0.2% [Lefurgy’11]
The banks of 96 NBTI aging sensors occupy less than 0.01% of the core's area [Singh’11]
Online Utilization of Guardbanding

The control system tunes the clock frequency through an online model-based rule.

1. Fine-grained granularity of instruction-by-instruction monitoring and adaptation that uses signals of PATV sensors from individual FUs
2. Coarse-grained granularity of kernel-level monitoring uses a representative PATV sensors for the entire execution stage of pipeline
Kernel-level monitoring improves throughput by 70% from P to PATV sensors. Target TER=0

Instruction-level monitoring improves throughput by 1.8-2.1X.
3. CHANGE CODE

COMPILE FOR HIGHER INSTRUCTION THROUGHPUT, LOWER POWER

Use Instruction Vulnerabilities to Generate Better Code, Call/Returns
Example: Procedure Hopping in Clustered CPU, Each core with its voltage domain

- Statically characterize procedure for PLV
- A core increases voltage if monitored delay is high
- A procedure hops from one core to another if its voltage variation is high
- Less 1% cycle overhead in EEMBC.

\[ V_{DD} = 0.81V \]

\[ V_{DD} = 0.99V \]

VA-\(V_{DD}\)-Hopping = (0.81V, 0.99V)
The code is easily accessible via the shared-L1 I$.
The data and parameters are passed through the shared stack in TCDM.
A procedure hopping information table (PHIT) keeps the status for a migrated procedure.
Consider shared 8-FPU 16-core architectures

COORDINATE: ADAPTATION TO PROPAGATE ERRORS TOWARDS APPLICATION
Accurate, Approximate Operating Modes

Modeled after STM P2012 16-core machine

- **Accurate mode**: every pipeline uses (with 3.8% area overhead)
  - EDS circuit sensors to detect any timing errors, ECU to correct errors using multiple-issue operation replay mechanism (without changing frequency)
Accuracy-Configurable Architecture

- In the approximate mode
  - Pipeline disables the EDS sensors on the less significant $N$ bits of the fraction where $N$ is reprogrammable through a memory-mapped register.
  - The sign and the exponent bits are always protected by EDS.
  - Thus pipeline ignores any timing error below the less significant $N$ bits of the fraction and save on the recovery cost.

- Switching between modes disables/enables the error detection circuits partially on $N$ bits of the fraction $\rightarrow$ FP pipeline can efficiently execute subsequent interleaved accurate or approximate software blocks.
Fine-grain Interleaving Possible Through Coordination and Controlled Approximation

**Architecture:** accuracy-reconfigurable FPUs that are shared among tightly-coupled processors and support online FPV characterization

**Compiler:** OpenMP pragmas for approximate FP computations; profiling technique to identify tolerable error significance and error rate

**Runtime:** Scheduler utilizes FPV metadata and *promotes* FPUs to accurate mode, or *demotes* them to approximate mode depending upon the code region requirements.

Either ignore the timing errors (in approximate regions) or reduce frequency of errors by assigning computations to correctible hardware resources for a cost.

Ensure safety of error ignorance through a set of rules.
FP Vulnerability Dynamically Monitored and Controlled by ECU

- % of cycles with timing errors as reported by EDS sensors captured as FPV metadata
- Metadata is visible to the software through memory-mapped registers.
- Enables runtime scheduler to perform on-line selection of best FP pipeline candidates
  - Low FPV units for accurate blocks, or steer error without correction to application.
OpenMP Compiler Extension

```c
#pragma omp accurate
structed-block
#pragma omp approximate [clause]
structured-block

#pragma omp parallel
{
#pragma omp accurate
#pragma omp for
for (i=K/2; i < (IMG_M-K/2); ++i) {
    // iterate over image
    for (j=K/2; j < (IMG_N-K/2); ++j) {
        float sum = 0;
        int ii, jj;
        for (ii = -K/2; ii <= K/2; ++ii) {
            // iterate over kernel
            for (jj = -K/2; jj <= K/2; ++jj) {
                float data = in[i+ii][j+jj];
                float coef = coeffs[ii+K/2][jj+K/2];
                float result;
                #pragma omp approximate error_significance_threshold(20)
                {
                    result = data * coef;
                    sum += result;
                }
            }
        }
        out[i][j] = sum/scale;
    }
}
```

Code snippet for Gaussian filter utilizing OpenMP variability-aware directives
The Expedition

ARTIFACTS DRIVE UNDERSTANDING
Expedition Artifacts: Molecule, ERSA, Cooper
Expedition Experimental Platforms & Artifacts: VarEMU VMM, ERSA, Molecule

Virtual Machine
- Instruction disassembly & translation
  - Cycle & Time Accounting
- VarEMU
  - Energy Accounting
  - Aging Model
- Virtual Hardware Device
  - Fault Model
  - Power Model

User + Software Monitor

Instrumented Hardware
- Performance
- Observability
- Control
- Cost

- Instrumented Hardware
  - ✔
- HW-In-The Loop Sim
  - ✔
- Gate/RTL-Level Sim
  - ✔
- VarEMU
  - ✔

Ming the Merciless
Red Cooper
ERSA@BEE3
Molecule
Red Cooper Testbed

- Customized chip with processor + speed/leakage sensors
- Testbed board to finish the sensor feedback loop on board
- Used in building a duty-cycled OS based on variability sensors
**Ferrari Chip: Closing Loop On-Chip**

- **On-Chip Sensors**
  - Memory mapped i/o and control
  - Leakage sensors, DDROs, temperature sensors, reliability sensors
- **Better support for OS and software**

Available since April 2013
Sense-and-Adapt Fundamentally Alters The Stack

- Machines that consist of parts with variations in performance, power and reliability
- Machines that incorporate sensing circuits
- Machines w/ interfaces to change ongoing computation & structures
- New machine models: QOS or Relaxed Reliability parts.
Thank You!

Rajesh K. Gupta

Nikil Dutt, UCI
Punit Gupta, UCLA
Mani Srivastava, UCLA
Steve Swanson, UCSD
Lara Dolecek, UCLA
Subhashish Mitra, Stanford

YY Zhou, UCSD
Tajana Rosing, UCSD
Alex Nicolau, UCI
Ranjit Jhala, UCSD
Sorin Lerner, UCSD
Rakesh Kumar, UIUC
Dennis Sylvester, UMich

Yuvraj Agrawal, CMU
Lucas Wanner, UCLA

http://variability.org