Variability-Resistant Software Through Improved Sensing & Modeling: 
Compiler Directed Strategies

Rajesh K. Gupta
Outline

• Why variability?
• Expedition’s view of UNO Machines
  – Sensors, Circuits, Instructions, Procedures, Tasks
  – Error rates, vulnerabilities, classifications
• Between sense & adapt and model & predict
  – Compile time optimization
  – Runtime adaptive guardbanding
• WIP results and summary.

Caveats: A limited view (entirely work by Abbas Rahimi, UCSD)
A very Expedition-centric view, not comprehensive, or even representative of Expeditions.
Variability is about Scale and Cost

- Variability in transistor characteristics is a major challenge in nanoscale CMOS, **PVTA**
  - Static **Process** variation: effective transistor channel length and threshold voltage
  - Dynamic variations: **Temperature** fluctuations, supply **Voltage droops**, and device **Aging** (NBTI, HCI)
- To handle variations → designers use conservative **guardbands** → loss of operational efficiency 😞
The real effect of variability is uncertainty

- **Two dimensions**
  - \{Spatial, Temporal, Dynamic\} x \{Deterministic, Stochastic\}
- **Spatial**
  - Manufacturing process variations, random defects
  - Affect yield right after production
- **Temporal**
  - Aging effects (HCI, NBTI, Soft Breakdown,...)
  - EM, TDDB, Corrosion,...
- **Dynamic**
  - Workload, temperature variations, EMI events
  - How the IC is used.

Deterministic/Stochastic: function of how physics is captured.
Temporal and Functional Uncertainties

• Temporal uncertainties are quite familiar to real-time systems community
  – Measures that span architectural simplification to OS simplification, structuring computation as precise and imprecise and combining with real-time OS models (FG/BG).

• PL: from performance to correctness to reliability

• Lately PL community has taken on fault-tolerant computations
  – How to avoid BSD? Decompose, Calibrate, Acceptance Tests
  – Probabilistic Accuracy Bound & Early Phase Termination, [Rinard, ICS 2006]
  – Principled Approximation [Baek/Chilimbi MSR 2009]

• Programmer approximates expensive functions, build a model of QoS loss by the approximation during calibration phase
  – Use model during operational phase to save energy by an adaptation function that monitors runtime behavior.
Figure 11. The tradeoff between QoS loss and the improvement in performance and energy consumption of Eon.

Figure 13. The tradeoff between QoS loss and the improvement in performance and energy consumption of CGA.

Figure 15. The tradeoff between QoS loss and the improvement in performance and energy consumption of DFT.
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, ...
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

- Sensors

- Models

Variability manifestations:
- faulty cache bits
- delay variation
- power variation
UnO Computing Machines: Taxonomy of Underdesign

- Parametric Underdesign
- Functional Underdesign

- Errored Operation
- Errorfree Operation

Nominal Design

Manufacturing

Performance Constraints

Manufactured Die

Die Specific Adaptation

Hardware Characterization Tests

Signature Burn In

Manufactured Die With Stored Signatures

Puneet Gupta/UCLA
Task Ingredients:
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
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).

Monitor manifestations from instructions levels to task levels.
Methodology

- Characterize effects of Dynamic Voltage and Temperature Variation
- Estimate their effects on instruction executions
  - Instruction-level Vulnerability (ILV)
  - Sequence-level Vulnerability (SLV)
- Classify instructions, and sequences of instructions
- Use ILV, SLV
  - Compile time optimization
  - Runtime adaptive guardbanding
Characterize

• Characterize LEON3 in 65nm TSMC across full range of operating conditions: (-40°C−125°C, 0.72V−1.1V)

• Dynamic variations contain both HF and LF components locally as well as across the die.

Dynamic variations cause the critical path delay to increase by a factor of \(6.1\times\).
One First Challenge: How do we make the leap to Software?

WAIT! DID WE MISS A STEP?
Connecting the dots: Delay and Pipestages

Observe:
The *execute* and *memory* parts are sensitive to V/T variations, and also exhibit a large number of critical paths in comparison to the rest of processor.

Hypothesis:
We anticipate that the instructions that significantly exercise the *execute* and *memory* stages are likely to be more vulnerable to V/T variations→ Instruction-level Vulnerability (ILV)
Method for ISA-level & Sequence-level Characterization

- For SPARC V8 instructions (V, T, F) are varied and
  - $ILV_i$ is evaluated for every instruction, with random operands
  - $SLV_i$ is evaluated for a high-frequent sequence, of instructions
Generate ILV, SLV “Metadata”

- The ILV (SLV) for each instruction_i (sequence_i) at every operating condition is quantified:

\[
ILV(i,V,T,\text{cycle}_\text{time}) = \frac{1}{N_i} \sum_{j=1}^{N_i} \text{Violation}_j \\
SLV(i,V,T,\text{cycle}_\text{time}) = \frac{1}{M_i} \sum_{j=1}^{M_i} \text{Violation}_j
\]

\[
\text{Violation}_j = \begin{cases} 
1 & \text{If any stage violates at cycle}_j \\
0 & \text{otherwise}
\end{cases}
\]

– where \( N_i (M_i) \) is the total number of clock cycles in Monte Carlo simulation of instruction_i (sequence_i) with random operands.

– \( \text{Violation}_j \) indicates whether there is a violated stage at clock cycle_j or not.

- ILV_i (SLV_i) defines as the total number of violated cycles over the total simulated cycles for the instruction_i (sequence_i).

Now, I am going to make a jump over characterization data...
1 Classify Instructions in 3 Classes

**ILV at 0.88V, while varying temperature:**

<table>
<thead>
<tr>
<th>(V, T)</th>
<th>(0.88V, -40°C)</th>
<th>(0.88V, 0°C)</th>
<th>(0.88V, 125°C)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cycle time (ns)</strong></td>
<td>1</td>
<td>1.02</td>
<td>1.06</td>
</tr>
<tr>
<td><strong>Logical &amp; Arithmetic</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>and</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>or</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sll</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sra</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>srl</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sub</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>xor</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Mem</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>load</td>
<td>1</td>
<td>0.824</td>
<td>0</td>
</tr>
<tr>
<td>store</td>
<td>1</td>
<td>0.847</td>
<td>0</td>
</tr>
<tr>
<td><strong>Mul. &amp; Div.</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mul</td>
<td>1</td>
<td>0.996</td>
<td>0.064</td>
</tr>
<tr>
<td>div</td>
<td>1</td>
<td>0.991</td>
<td>0.989</td>
</tr>
</tbody>
</table>

- Instructions are partitioned into three main classes: (i) Logical & arithmetic; (ii) Memory; (iii) Multiply & divide.
- The 1st class shows an abrupt behavior when the clock cycle is slightly varied, mainly because the path distribution of the exercised part by this class is such that most of the paths have the same length, then we have all-or-nothing effect, which implies that either all instructions within this class fail or all make it.
2 Check them across temperature

ILV at 0.72V, while varying temperature:

<table>
<thead>
<tr>
<th>Corners</th>
<th>(0.72V, -40°C)</th>
<th>(0.72V, 0°C)</th>
<th>(0.72V, 125°C)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cycle time (ns)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4.10</td>
<td>4.12</td>
<td>4.14</td>
</tr>
<tr>
<td>add</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>and</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>or</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sll</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sra</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>srl</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sub</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>xor</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>xnor</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>load</td>
<td>1</td>
<td>0.823</td>
<td>0.823</td>
</tr>
<tr>
<td>store</td>
<td>1</td>
<td>0.847</td>
<td>0.847</td>
</tr>
<tr>
<td>mul</td>
<td>1</td>
<td>0.995</td>
<td>0.995</td>
</tr>
<tr>
<td>div</td>
<td>1</td>
<td>0.995</td>
<td>0.995</td>
</tr>
</tbody>
</table>

- All instruction classes act similarly across the wide range of operating conditions: as the cycle time increases gradually, the ILV becomes 0, firstly for the 1st class, then for the 2nd class, and finally for the 3rd class.
- For every operating conditions

\[
\text{ILV (3rd Class)} \geq \text{ILV (2nd Class)} \geq \text{ILV (1st Class)}
\]
### 3 Classify Instruction Sequences

SLV at (0.81V, 125C)

<table>
<thead>
<tr>
<th>CT (ns)</th>
<th>Seq1</th>
<th>Seq2</th>
<th>Seq3</th>
<th>Seq4</th>
<th>Seq5</th>
<th>Seq6</th>
<th>Seq7</th>
<th>Seq8</th>
<th>Seq9</th>
<th>Seq10</th>
<th>Seq11</th>
<th>Seq12</th>
<th>Seq13</th>
<th>Seq14</th>
<th>Seq15</th>
<th>Seq16</th>
<th>Seq17</th>
<th>Seq18</th>
<th>Seq19</th>
<th>Seq20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.26</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1.27</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1.28</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1.29</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1.30</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1.31</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1.32</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1.33</td>
<td>0.878</td>
<td>0.811</td>
<td>0.881</td>
<td>0.880</td>
<td>0.884</td>
<td>0.892</td>
<td>0.877</td>
<td>0.859</td>
<td>0.879</td>
<td>0.758</td>
<td>0.883</td>
<td>0.883</td>
<td>0.811</td>
<td>0.811</td>
<td>0.952</td>
<td>0.811</td>
<td>0.805</td>
<td>0.810</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1.34</td>
<td>0.366</td>
<td>0.811</td>
<td>0.515</td>
<td>0.512</td>
<td>0.393</td>
<td>0.429</td>
<td>0.859</td>
<td>0.03</td>
<td>0.403</td>
<td>0.407</td>
<td>0.811</td>
<td>0.811</td>
<td>0.811</td>
<td>0.811</td>
<td>0.811</td>
<td>0.811</td>
<td>0.805</td>
<td>0.810</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1.35</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- The top 20 high-frequent sequences (Seq1-Seq20) are extracted from 80 Billion dynamic instructions of 32 benchmarks.

- Sequences are classified into two classes based on their similarities in SLV values:
  - **Class I** (Seq20) only consists of the arithmetic/logical instructions.
  - **Class II** (Seq1-Seq19) is a mixture of all types of instructions including the memory, arithmetic/logical, and control instructions.
### Classification of Sequence of Instructions (2/3)

#### SLV at (0.81V, -40°C).

The same trend with 165°C temperature variations.

| CT (ns) | Seq1 | Seq2 | Seq3 | Seq4 | Seq5 | Seq6 | Seq7 | Seq8 | Seq9 | Seq10 | Seq11 | Seq12 | Seq13 | Seq14 | Seq15 | Seq16 | Seq17 | Seq18 | Seq19 | Seq20 |
|---------|------|------|------|------|------|------|------|------|------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 1.36    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 0.475 |
| 1.37    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 0     |
| 1.38    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 0     |
| 1.39    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 0     |
| 1.40    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 0     |
| 1.41    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 0     |
| 1.42    | 0.878| 0.811| 0.881| 0.880| 0.884| 0.892| 0.877| 0.859| 0.988| 0.758  | 0.882  | 0.883  | 0.811  | 0.811  | 0.815  | 0.870  | 0.811  | 0.807  | 0.810  |
| 1.43    | 0.01 | 0.01 | 0.479| 0.396| 0.06 | 0.04 | 0.01 | 0.01 | 0.901| 0.01   | 0.01   | 0.01   | 0.811  | 0.811  | 0.811  | 0.811  | 0.810  | 0.805  | 0.131  |
| 1.44    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0     |

#### (V,T) = (0.81V, 125°C)

<table>
<thead>
<tr>
<th>CT (ns)</th>
<th>Class II</th>
<th>Class I</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.26</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1.27</td>
<td>1</td>
<td>0.69</td>
</tr>
<tr>
<td>1.28</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1.29</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1.30</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1.31</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1.32</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1.33</td>
<td>0.81</td>
<td>0</td>
</tr>
<tr>
<td>1.34</td>
<td>0.81</td>
<td>0</td>
</tr>
<tr>
<td>1.35</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

#### (V,T) = (0.72V, 125°C)

<table>
<thead>
<tr>
<th>CT (ns)</th>
<th>Class II</th>
<th>Class I</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.78</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1.79</td>
<td>1</td>
<td>0.58</td>
</tr>
<tr>
<td>1.80</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1.81</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1.82</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1.83</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1.84</td>
<td>1.81</td>
<td>0</td>
</tr>
<tr>
<td>1.85</td>
<td>0.13</td>
<td>0</td>
</tr>
<tr>
<td>1.86</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1.87</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

For every operating conditions:

**SLV (Class II) ≥ SLV (Class I)**

Sequences in *Class II* need higher guardbands compared to *Class I*, because in addition of ALU's critical paths, the critical paths of memory are activated (for the load/store instructions) as well as the critical paths of integer code conditions (for the control instructions).
For every operating conditions:

ILV (3\textsuperscript{rd} Class) ≥ ILV (2\textsuperscript{nd} Class) ≥ ILV (1\textsuperscript{st} Class)

SLV (\textit{Class II}) ≥ SLV (\textit{Class I})

ILV and SLV classification for integer SPARC V8 ISA.
I. Error-tolerant Applications
   - Duplication of critical instructions
   - Satisfying the fidelity metric

II. Error-intolerant Application
   - Increasing the percentage of the sequences of Class I, i.e., increasing the number of arithmetic instructions with regard to the memory and control flow instructions, e.g., through loop unrolling technique

• Adaptive clock scaling for each class of sequences mitigates the conservative inter- and intra-corner guardbanding.

• At the runtime, in every cycle, the PLUT module sends the desired frequency to the adaptive clocking circuit utilizing the characterized SLV metadata of the current sequence and the operating condition monitored by CPM.
• Applying the loop unrolling produces a longer chain of ALU instructions, and as a result the percentage of sequences of *Class1* is increased up to 41% and on average 31%.

• Hence, the adaptive guardbanding benefits from this compiler transformation technique to further reduce the guardband for sequences of *Class1*. 
Effectiveness of Adaptive Guardbanding

• Using online SLV coupled with offline compiler techniques enables the processor to achieve $1.6 \times$ average speedup for intolerant applications, compared to recent work [Hoang’11], by adapting the cycle time for dynamic variations (inter-corner) and different instruction sequences (intra-corner).

• Adaptive guardbanding achieves up to $1.9 \times$ performance improvement for error-tolerant (probabilistic) applications in comparison to the traditional worst-case design.

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.

\[
\begin{align*}
V_{DD} &= 0.81V \\
V_{DD} &= 0.99V \\
\text{VA-V}_{DD}\text{-Hopping} &= (0.81V, 0.99V)
\end{align*}
\]
HW/SW Collaborative Architecture to Support Intra-cluster Procedure Hopping

- 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.
NOT SURE HOW FAR WE CAN PUSH THIS SENSING. REMEMBER ILP?
The model takes into account:
1. PVTA parameter variations
2. Clock frequency
3. Physical details of Placed-and-Routed FUs in 45nm TSMC technology

- Analyzed FUs:
  - 10 32-bit integer
  - 15 single precision floating-point (fully compatible with the IEEE 754 standard)
- A full permutation of PVTA parameters and clock frequency are applied.

For each FU_i working with t_{clk} and a given PVTA variations, we defined Timing Error Rate (TER):

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

<table>
<thead>
<tr>
<th>Parameter</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 (σ_{WID})</td>
<td>0%</td>
<td>9.6%</td>
<td>3.2%</td>
<td>4</td>
</tr>
<tr>
<td>Aging (ΔV_{th})</td>
<td>0mV</td>
<td>100mV</td>
<td>25mV</td>
<td>5</td>
</tr>
<tr>
<td>t_{clk}</td>
<td>0.2ns</td>
<td>5.0ns</td>
<td>0.2ns</td>
<td>25</td>
</tr>
</tbody>
</table>
We used Supervised learning (linear discriminant analysis) to generate a parametric model at the level of FU that relates PVTA parameters variation and $t_{clk}$ to classes of TER.

On average, for all FUs the resubstitution error is $0.036$, meaning the models classify nearly all data correctly.

For extra characterization points, the model makes correct estimates for $97\%$ of out-of-sample data. The remaining $3\%$ is misclassified to the high-error rate class, $C_H$, thus will have safe guardband.
During design time the delay of the FP adder has a large uncertainty of [0.73ns,1.32ns], since the actual values of PVTA parameters are unknown.
The question is that mix of monitors that would be useful?

Sensor overheads:

✓ *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]

• 24% (P_sensor)
• 28% (PA_sensors),
• 44% (PATV_sensors)
Online Utilization of HFG

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

- To support fast controller's computation, the parametric model generates distinct Look Up Tables (LUTs) for every FUs.

- We apply HFG to architecture at two granularities
  1. Fine-grained granularity of instruction-by-instruction monitoring and adaptation that signals of PATV sensors come from individual FUs
  2. Coarse-grained granularity of kernel-level monitoring uses a representative PATV sensors for the entire execution stage of pipeline
1. At kernel-level monitoring, on average, the throughput increases by 70%, when the PE moves from only P_sensor to PATV_sensors scenario. The target TER is set to “0” in preference to the error-intolerant applications.

2. Instruction-by-instruction monitoring and adaptation improves the throughput by $1.8 \times -2.1 \times$ depends to the PATV sensors configuration and kernel's instructions.
Thank You!

The Variability Expedition
http://variability.org

A NSF Expeditions in Computing Project