Voltage Overscaling Algorithms for Energy-Efficient Workflow Computations With Timing Errors

Aurélien Cavelan\textsuperscript{1}, Yves Robert\textsuperscript{1,2}, Hongyang Sun\textsuperscript{1} and Frédéric Vivien\textsuperscript{1}

1. ENS Lyon & INRIA, France
2. University of Tennessee Knoxville, USA

aurelien.cavelan@ens-lyon.fr

Green Days – Toulouse
March 17, 2015
Dynamic Power Consumption

One can use *Dynamic Voltage and Frequency Scaling (DVFS)* to reduce power consumption.
Dynamic Power Consumption

One can use *Dynamic Voltage and Frequency Scaling (DVFS)* to reduce power consumption.

\[
\text{Power} = \alpha fV^2
\]

- $\alpha$ the effective capacitance
- $f$ the frequency
- $V$ the operating voltage
Dynamic Power Consumption

One can use *Dynamic Voltage and Frequency Scaling (DVFS)* to reduce power consumption.

\[ \text{Power} = \alpha fV^2 \]

- \(\alpha\) the effective capacitance
- \(f\) the frequency
- \(V\) the operating voltage

⇒ Voltage has a quadratic impact on the dynamic power.
The Easy Way

We target energy consumption only, not time.

For any frequency value, there is a threshold voltage:
The Easy Way

We target energy consumption only, not time.

For any frequency value, there is a threshold voltage:

1. Find the frequency that minimizes energy consumption
The Easy Way

We target energy consumption only, not time.

For any frequency value, there is a threshold voltage:

1. Find the frequency that minimizes energy consumption
2. Decrease the voltage to \textit{threshold voltage}
We target energy consumption only, not time.

For any frequency value, there is a threshold voltage:

1. Find the frequency that minimizes energy consumption
2. Decrease the voltage to *threshold voltage*

Can we do better?
Threshold Voltage

Figure: Set of voltages of a FPGA multiplier block and the associated error probabilities measured on random inputs at 90MHz and 27°C
Timing Errors

**Definition**

The results of some logic gates could be used before their output signals reach their final values.

- Occur when $V_{DD} < V_{TH}$
- *Deterministic* but *unpredictable*
- Induce Silent Data Corruptions (SDC)
Timing Errors

Definition

The results of some logic gates could be used before their output signals reach their final values.

- Occur when $V_{DD} < V_{TH}$
- Deterministic but unpredictable
- Induce Silent Data Corruptions (SDC)

Unlike lightning, timing errors always strike twice
Timing Errors

Definition

The results of some logic gates could be used before their output signals reach their final values.

- Occur when $V_{DD} < V_{TH}$
- Deterministic but unpredictable
- Induce Silent Data Corruptions (SDC)

Unlike lightning, timing errors always strike twice

Silent errors are detected only when the corrupt data is activated
Two Approaches

Near-Threshold Computing ($V_{dd} \approx V_{th}$)

- Used in NTC circuits (hardware)
- Almost safe 😊
- Great energy savings 😊
Two Approaches

**Near-Threshold Computing (\(V_{dd} \approx V_{th}\))**
- Used in NTC circuits (hardware)
- Almost *safe* 😊
- Great energy savings 😊

**Voltage Overscaling (\(V_{dd} < V_{th}\))**
- Even more energy savings 😊
- Purely software-based approach 😊
- Generate timing errors 😞
- Require a verification mechanism 😞
- Require probabilities of failure for the platform 😞
Is it possible to obtain the (correct) result of a computation for a lower energy budget than that of the best DVFS / NTC solution?
Outline

1. Introduction
2. Theoretical Approach
3. Simulations
4. Conclusion
Model Assumptions

Consider a task and a set of voltages $\mathcal{V}$:

<table>
<thead>
<tr>
<th>Voltages</th>
<th>$V_1$</th>
<th>$V_2$</th>
<th>...</th>
<th>$V_m = V_{TH}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Pr(V_\ell\text{-fail})$</td>
<td>$p_1$</td>
<td>$p_2$</td>
<td>...</td>
<td>$p_m = 0$</td>
</tr>
<tr>
<td>Cost</td>
<td>$c_1$</td>
<td>$c_2$</td>
<td>...</td>
<td>$c_m$</td>
</tr>
</tbody>
</table>
Model Assumptions

Consider a task and a set of voltages $\mathcal{V}$:

<table>
<thead>
<tr>
<th>Voltages</th>
<th>$V_1$</th>
<th>$V_2$</th>
<th>$\cdots$</th>
<th>$V_m = V_{TH}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Pr(V_\ell\text{-fail})$</td>
<td>$p_1$</td>
<td>$p_2$</td>
<td>$\cdots$</td>
<td>$p_m = 0$</td>
</tr>
<tr>
<td>Cost</td>
<td>$c_1$</td>
<td>$c_2$</td>
<td>$\cdots$</td>
<td>$c_m$</td>
</tr>
</tbody>
</table>

Remember: timing errors always strike twice.
Model Assumptions

Consider a task and a set of voltages $\mathcal{V}$:

<table>
<thead>
<tr>
<th>Voltages</th>
<th>$V_1$</th>
<th>$V_2$</th>
<th>...</th>
<th>$V_m = V_{TH}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(V_\ell$-fail)</td>
<td>$p_1$</td>
<td>$p_2$</td>
<td>...</td>
<td>$p_m = 0$</td>
</tr>
<tr>
<td>Cost</td>
<td>$c_1$</td>
<td>$c_2$</td>
<td>...</td>
<td>$c_m$</td>
</tr>
</tbody>
</table>

Remember: timing errors always strike twice.

- When an error strikes, a higher voltage must be used
- Switching from voltage $V_\ell$ to $V_h$ incurs a cost $o_{\ell,h}$
- Execution at $V_{TH}$ always succeeds
Consider a task and a set of voltages $\mathcal{V}$:

<table>
<thead>
<tr>
<th>Voltages</th>
<th>$V_1$</th>
<th>$V_2$</th>
<th>...</th>
<th>$V_m = V_{TH}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{P}(V_\ell\text{-fail})$</td>
<td>$p_1$</td>
<td>$p_2$</td>
<td>...</td>
<td>$p_m = 0$</td>
</tr>
<tr>
<td>Cost</td>
<td>$c_1$</td>
<td>$c_2$</td>
<td>...</td>
<td>$c_m$</td>
</tr>
</tbody>
</table>

Remember: timing errors always strike twice.

- When an error strikes, a higher voltage must be used
- Switching from voltage $V_\ell$ to $V_h$ incurs a cost $o_{\ell,h}$
- Execution at $V_{TH}$ always succeeds

How to compute the probability of failure?
Consider a task and a set of voltages $V$:

<table>
<thead>
<tr>
<th>Voltages</th>
<th>$V_1$</th>
<th>$V_2$</th>
<th>$\ldots$</th>
<th>$V_m = V_{TH}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(V_{\ell\text{-fail}})$</td>
<td>$p_1$</td>
<td>$p_2$</td>
<td>$\ldots$</td>
<td>$p_m = 0$</td>
</tr>
<tr>
<td>Cost</td>
<td>$c_1$</td>
<td>$c_2$</td>
<td>$\ldots$</td>
<td>$c_m$</td>
</tr>
</tbody>
</table>

Remember: timing errors always strike twice.

- When an error strikes, a higher voltage *must* be used.
- Switching from voltage $V_{\ell}$ to $V_h$ incurs a cost $o_{\ell,h}$.
- Execution at $V_{TH}$ always succeeds.

How to compute the probability of failure?

The optimal sequence of voltages?
Property of Timing Errors

1. Given an operation and an input $I$, there exists a threshold voltage $V_{\text{TH}}(I)$:
   - $V < V_{\text{TH}}(I)$ will always lead to an incorrect result
   - $V \geq V_{\text{TH}}(I)$ will always lead to a successful execution
Property of Timing Errors

1. Given an operation and an input $I$, there exists a *threshold voltage* $V_{\text{TH}}(I)$:
   - $V < V_{\text{TH}}(I)$ will always lead to an incorrect result
   - $V \geq V_{\text{TH}}(I)$ will always lead to a successful execution

2. Given an operation and a voltage $V \in \mathcal{V}$:
   - $\mathcal{I}$ denotes the set of all possible inputs
   - $\mathcal{I}_f(V) \subseteq \mathcal{I}$ is the set of inputs that will fail at voltage $V$
   - Failure probability is computed as $p_V = |\mathcal{I}_f(V)|/|\mathcal{I}|$
   - For any two voltages $V_1 \geq V_2$, we have $\mathcal{I}_f(V_1) \subseteq \mathcal{I}_f(V_2)$
Property of Timing Errors

1. Given an operation and an input $I$, there exists a *threshold voltage* $V_{\text{TH}}(I)$:
   - $V < V_{\text{TH}}(I)$ will always lead to an incorrect result
   - $V \geq V_{\text{TH}}(I)$ will always lead to a successful execution

2. Given an operation and a voltage $V \in \mathcal{V}$:
   - $\mathcal{I}$ denotes the set of all possible inputs
   - $\mathcal{I}_f(V) \subseteq \mathcal{I}$ is the set of inputs that will fail at voltage $V$
   - Failure probability is computed as $p_V = \frac{|\mathcal{I}_f(V)|}{|\mathcal{I}|}$
   - For any two voltages $V_1 \geq V_2$, we have $\mathcal{I}_f(V_1) \subseteq \mathcal{I}_f(V_2)$

$$
\mathbb{P}(V_{\ell}\text{-fail} \mid V_0, V_1, \ldots, V_{\ell-1}\text{-fail}) = \frac{|\mathcal{I}_f(V_{\ell})|/|\mathcal{I}|}{|\mathcal{I}_f(V_{\ell-1})|/|\mathcal{I}|} = \frac{p_\ell}{p_{\ell-1}}
$$
Energy Consumption of a Single Task

Consider a sequence $L$ of $k$ voltages $V_1 < V_2 < \cdots < V_k = V_{TH}$,

Execution starts at voltage $V_1$:

1. In case of success, return the result!
2. In case of failure, go to next (higher) voltage

\[
E(L) = c_1 + p_1(o_1, 2) + c_2 + p_2(o_2, 3) + \cdots + p_{k-1}(o_{k-1}, k) + c_k
\]

We generalize:

\[
E(L) = c_1 + k \sum_{\ell=2}^{k} p_{\ell-1}(o_{\ell-1}, \ell) \tag{1}
\]
Energy Consumption of a Single Task

Consider a sequence $L$ of $k$ voltages $V_1 < V_2 < \cdots < V_k = V_{TH}$.

Execution starts at voltage $V_1$:

1. In case of success, return the result!
2. In case of failure, go to next (higher) voltage

$$E(L) = c_1 + p_1 \left( o_{1,2} + c_2 + \frac{p_2}{p_1} \left( o_{2,3} + c_3 + \cdots \frac{p_{k-1}}{p_{k-2}}(o_{k-1,k} + c_k) \right) \right)$$

$$= c_1 + p_1(o_{1,2} + c_2) + p_2(o_{2,3} + c_3) + \cdots + p_{k-1}(o_{k-1,k} + c_k)$$
Energy Consumption of a Single Task

Consider a sequence $L$ of $k$ voltages $V_1 < V_2 < \cdots < V_k = V_{\text{TH}}$.

Execution starts at voltage $V_1$:

1. In case of success, return the result!
2. In case of failure, go to next (higher) voltage

$$E(L) = c_1 + p_1 \left( o_{1,2} + c_2 + \frac{p_2}{p_1} \left( o_{2,3} + c_3 + \cdots \frac{p_{k-1}}{p_{k-2}}(o_{k-1,k} + c_k) \right) \right)$$

$$= c_1 + p_1(o_{1,2} + c_2) + p_2(o_{2,3} + c_3) + \cdots + p_{k-1}(o_{k-1,k} + c_k)$$

We generalize:

$$E(L) = c_1 + \sum_{\ell=2}^{k} p_{\ell-1} (o_{\ell-1,\ell} + c_\ell) \quad (1)$$
Optimal Sequence of Voltages

Theorem

To minimize the expected energy consumption for a single task, the optimal sequence of voltages to execute the task with a preset voltage $V_p \in \mathcal{V}$ of the system can be obtained by dynamic programming with complexity $O(k^2)$. 
Optimal Sequence of Voltages

Theorem

To minimize the expected energy consumption for a single task, the optimal sequence of voltages to execute the task with a preset voltage $V_p \in \mathcal{V}$ of the system can be obtained by dynamic programming with complexity $O(k^2)$.

\[
E(L_s^*) = c_s + \min_{s<\ell\leq k} \{ E(L^*_\ell) - c_\ell + p_s(o_s,\ell + c_\ell) \} \quad (2)
\]

and the optimal sequence starting with $V_s$ is $L_s^* = \langle V_s, L^*_{\ell'} \rangle$ where

\[
\ell' = \arg\min_{s<\ell\leq k} \{ E(L^*_\ell) + p_s o_s,\ell + (p_s - 1)c_\ell \}.
\]

The dynamic program is initialized with $E(L^*_k) = c_k$ and $L^*_k = \langle V_k \rangle$. 

Chain of Tasks

- Without switching cost: optimal sequence for one task can be used to execute each task.
Chain of Tasks

- **Without switching cost:** optimal sequence for one task can be used to execute each task.

- **With switching cost:**
  - After execution of a task, platform is left at voltage $V_e$
  - Optimal sequence starts at voltage $V_s$
  - Additional switching cost $o_{e,s}$ must be paid
  - Algorithm for one task is no longer optimal
Chain of Tasks

- **Without switching cost:** optimal sequence for one task can be used to execute each task.

- **With switching cost:**
  - After execution of a task, platform is left at voltage $V_e$
  - Optimal sequence starts at voltage $V_s$
  - Additional switching cost $o_{e,s}$ must be paid
  - Algorithm for one task is no longer optimal

**Theorem**

*To minimize the expected energy consumption for a linear chain of tasks, the optimal sequence of voltages to execute each task, given the terminating voltage of its preceding task (or given the preset voltage $V_p$ of the system for the first task), can be obtained by dynamic programming with complexity $O(nk^2)$.*
Outline

1. Introduction
2. Theoretical Approach
3. Simulations
4. Conclusion
Consider the blocked matrix multiplication $C = A \times B$.

**Application Workflow**

```plaintext
for i = 1 to $\lceil \frac{m}{b} \rceil$ do
  for j = 1 to $\lceil \frac{m}{b} \rceil$ do
    for k = 1 to $\lceil \frac{m}{b} \rceil$ do
      $C_{i,j} \leftarrow C_{i,j} + A_{i,k} \times B_{k,j}$
```

- $m$ denotes the matrix size
- $b$ denotes the block size
Consider the blocked matrix multiplication $C = A \times B$.

**Application Workflow**

\[
\begin{align*}
\text{for } i &= 1 \text{ to } \left\lfloor \frac{m}{b} \right\rfloor \text{ do} \\
\text{for } j &= 1 \text{ to } \left\lfloor \frac{m}{b} \right\rfloor \text{ do} \\
\text{for } k &= 1 \text{ to } \left\lfloor \frac{m}{b} \right\rfloor \text{ do} \\
C_{i,j} &\leftarrow C_{i,j} + A_{i,k} \times B_{k,j}
\end{align*}
\]

- $m$ denotes the matrix size
- $b$ denotes the block size

*ABFT* can be used to add per-block verification.
Algorithm Based Fault Tolerance (ABFT)

Let $e^T = [1, 1, \cdots, 1]$, we define

$$A^c := \begin{pmatrix} A \\ e^T A \end{pmatrix}, \quad B^r := \begin{pmatrix} B & Be \end{pmatrix}, \quad C^f := \begin{pmatrix} C & Ce \\ e^T C & e^T Ce \end{pmatrix}.$$

Where $A^c$ is the column checksum matrix, $B^r$ is the row checksum matrix and $C^f$ is the full checksum matrix.
Algorithm Based Fault Tolerance (ABFT)

Let $e^T = [1, 1, \cdots, 1]$, we define

$$A^c := \begin{pmatrix} A \\ e^T A \end{pmatrix}, \quad B^r := \begin{pmatrix} B & Be \end{pmatrix}, \quad C^f := \begin{pmatrix} C & Ce \\ e^T C & e^T Ce \end{pmatrix}. $$

Where $A^c$ is the column checksum matrix, $B^r$ is the row checksum matrix and $C^f$ is the full checksum matrix.

$$A^c \times B^r = \begin{pmatrix} A \\ e^T A \end{pmatrix} \times \begin{pmatrix} B & Be \end{pmatrix} = \begin{pmatrix} AB & ABe \\ e^T AB & e^T ABe \end{pmatrix} = \begin{pmatrix} C & Ce \\ e^T C & e^T Ce \end{pmatrix} = C^f$$
Consider the matrix multiplication as a chain of $n = \lceil \frac{m}{b} \rceil^3$ tasks.

**Time to Execute one Task**

- $t = \tau \cdot w / \eta$
  - $\tau = \frac{1}{f}$ time to do one cycle
  - $\eta = 0.8$ peak performance
- $w = b(b + 1)^2 + \sigma$
  - $\sigma = 8^3$ initialization overhead
  - $(b + 1)$ ABFT overhead
Matrix Parameters

Consider the matrix multiplication as a chain of $n = \lceil \frac{m}{b} \rceil^3$ tasks.

### Time to Execute one Task

- $t = \tau \cdot w / \eta$
  - $\tau = 1/f$ time to do one cycle
  - $\eta = 0.8$ peak performance
- $w = b(b + 1)^2 + \sigma$
  - $\sigma = 8^3$ initialization overhead
  - $(b + 1)$ ABFT overhead

### Failure Probabilities

Consider a set of voltages $\mathcal{V}$. For any voltage $V_\ell \in \mathcal{V}$

- $p_\ell = 1 - (1 - p_\ell^{(1)}/\gamma)^w$
  - $\gamma = \frac{\text{silent errors}}{\text{timing errors}}$
  - $p_\ell^{(1)}$ probability of timing error for one random operation
Platform Settings

From [1] for a FPGA at $f = 90$MHz and $27^\circ$C:

- Set of voltages
- Timing errors probabilities

**Dynamic Power Consumption**

- $P(V, f) = \alpha f V^2$
- We assume (wlog) $\alpha f = 1$
Platform Settings

From [1] for a FPGA at \( f = 90\text{MHz} \) and \( 27^\circ \text{C} \):

- Set of voltages
- Timing errors probabilities

**Dynamic Power Consumption**

\[
P(V, f) = \alpha f V^2
\]

We assume (wlog) \( \alpha f = 1 \)
Platform Settings

From [1] for a FPGA at $f = 90\text{MHz}$ and $27^\circ\text{C}$:

- Set of voltages
- Timing errors probabilities

**Dynamic Power Consumption**

$$P(V, f) = \alpha f V^2$$

We assume (wlog) $\alpha f = 1$

**Voltage Switching Cost**

$$o_{\ell, h} = \begin{cases} 
0, & \text{if } \ell = h \\
\beta \cdot \frac{|V_\ell - V_h|}{V_k - V_1} & \text{otherwise}
\end{cases}$$

$$\beta = o_{1, k}$$
N-Voltage: Baseline algorithm that applies NTC and always uses threshold voltage.

$DP_1$-detect & $DP_1$-correct: Optimal dynamic programming algorithms for a single task.

$DP_n$-detect & $DP_n$-correct: Optimal dynamic programming algorithms for a chain of tasks.

detect algorithms use ABFT for error detection.
correct algorithms use ABFT for detection and correction.
Figure: Failure probabilities for one operation and for one task under different block sizes and voltages.
**Simulations** (without switching cost)

**Figure:** Impact of $b$ and $\gamma$ on the expected energy consumption for zero voltage switching cost. Only the results for the $\text{DP}_n$-correct algorithm are shown.
Simulations (with switching cost)

Figure: Impact of $b$ and $\beta$ on the expected energy consumption. The voltage switching cost is equivalent to the energy consumed to multiply two $32 \times 32$ matrices at threshold voltage without overhead.
We use dynamic voltage overscaling to reduce power consumption.

Summary

- Software based approach
- Model for timing errors
- Optimal polynomial-time solution for a chain of tasks
- Simulations on matrix multiplication using ABFT
Conclusion

We use dynamic voltage overscaling to reduce power consumption.

Summary

- Software based approach
- Model for timing errors
- Optimal polynomial-time solution for a chain of tasks
- Simulations on matrix multiplication using ABFT

Original problem and encouraging results; needs further research.
We use dynamic voltage overscaling to reduce power consumption.

**Summary**

- Software based approach
- Model for timing errors
- Optimal polynomial-time solution for a chain of tasks
- Simulations on matrix multiplication using ABFT

**Future Work**

- Algorithms for other task graphs
- Additional simulations, emulations and experiments

Original problem and encouraging results; needs further research.
Thanks! 😊
D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. S. Kim, and K. Flautner.
Razor: circuit-level correction of timing errors for low-power operation.