SAND2016-3158PE

Sandia National Laboratories

Error-Corrected Computing

Erik P. DeBenedictis, Center for Computing Research, Sandia
Hans Zima, Jet Propulsion Laboratory, Caltech
Presentation at Georgia Tech, May 21, 2015

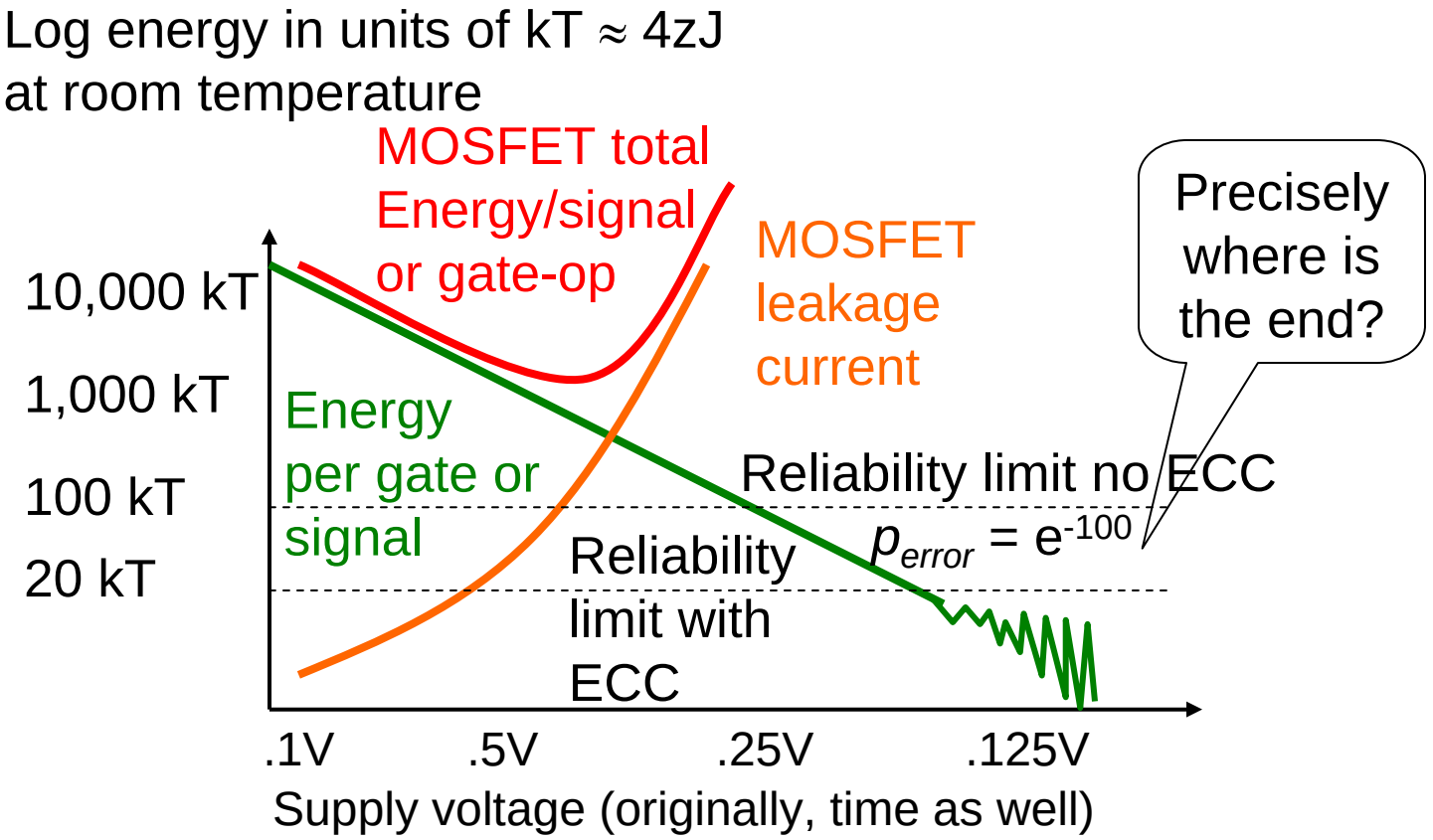
Approved for Unclassified Unlimited Release
Tracking number 275883

U.S. DEPARTMENT OF ENERGY | NNSA

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Future of CMOS

- Industry stuck in local minimum due to leakage current
- Scaling resumes someday to reliability limit (10x-100x energy)



At stake? Maybe one generation

- Scaling will not stop abruptly, but it will be stopped by an exponential rise in error rate with declining energy
- But how much energy efficiency improvement is possible if we can tolerate errors? Spreadsheet →
  - No ECC 71 kT
  - ECC scenarios 24 kT – 28 kT
  - 2:1 after overhead, +/-
- A trillion dollar question

Exascale reliability requirement						
100000	Gates-ops per floating point op where an error would cause a wrong answer					
1.00E+18	ops/second (definition of Exascale)					
60	seconds per minue					
60	minutes per hour					
24	hours per day					
365	days per year					
3	years for a computer's lifetime (before it becomes obsolete)					
9.46E+30	number of gate operations per lifetime where an error would cause a wrong answer					
71.33211	If we have Esignal equal this many kT's, error rate will be inverse of previous line					
Say an operation is this many gate-ops		1000	20000	1.00E+05	1.00E+06	1.00E+18
Steps in lifetime (serial and parallel)		9.46E+27	4.73E+26	9.46E+25	9.46E+24	9.46E+12
RRNS using system in Watson and Hastings						
Gate ops per residue (four non-redundant residue		250	5000	25000	250000	2.5E+17
perror target for exaflops over lifetime		1	1	1	1	1
perror per step		1.06E-28	2.11E-27	1.06E-26	1.06E-25	1.06E-13
perror per residue; 3 errors in a step must go und		7.02E-09	1.91E-08	3.26E-08	7.02E-08	7.02E-04
Es = this many kTs will meet reliability in line abov		24.30	26.30	27.37	28.90	47.33
Energy savings		2.94	2.71	2.61	2.47	1.51
However, we need 6 total residues, not 4		1.96	1.81	1.74	1.65	1.00
Additional beneficial factors						
Fixes Cosmic Ray hits						
Fixes weak and aging components						
Could support overclocking; i. e. catches an "excessive overclocking" error						

Outline

- Energy-reliability tradeoff
- Error correction for logic
- Redundant Residue Number System
- “Creepy” architecture
- Programming (assertion language)
- Integration with Processor-In-Memory-and-Storage (PIMS)
- PIMS programming

Plan for scaling from Theis and Solomon

Conventional Logic: Reduce the stored energy $\frac{1}{2}CV^2$. For conventional FETs, as V approaches a small multiple of kT/e, we must accept reduction in switching speed. New device concepts, discussed below, may allow more significant reduction in V and facilitate the reduction of stored energy towards kT. As thermal voltage fluctuations become significant, we must incorporate redundancy and error correction in the logic to keep the error rate in bounds. Refrigeration can reduce T, but in a power-constrained environment, this only makes economic sense if the total power needed to perform the computation is reduced, including the power for refrigeration.

$$p_{error} = \frac{1}{2}\mathrm{Erfc}[m/\sqrt{2}] \approx \exp(-E_{signal} / kT)$$

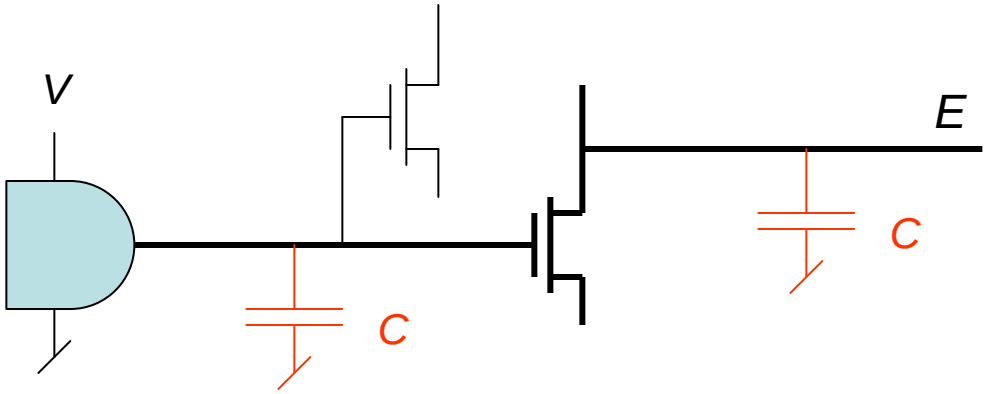
Since Johnson–Nyquist voltage noise is Gaussian with a standard deviation of V_n, a stored logic voltage of m standard deviations, or a stored energy of m^2kT, would be needed to achieve a reliability of $\frac{1}{2}\mathrm{Erfc}[m/\sqrt{2}]$. (Eight standard deviations give an error probability of $\sim 10^{-15}$.)

Both from Theis, Thomas N., and Paul M. Solomon. "In Quest of the" Next Switch": Prospects for Greatly Reduced Power Dissipation in a Successor to the Silicon Field-Effect Transistor." *Proceedings of the IEEE* 98.12 (2010): 2005-2014.

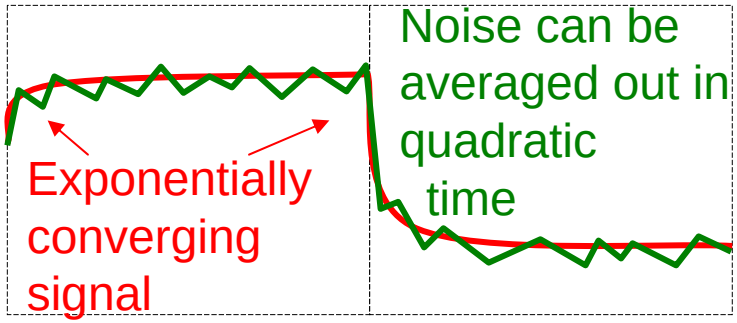
Circuit explanation

- Signal = $E_{signal} = CV^2$, where C is wire + fanout × C_{gate}
- Noise power is $2f$kT
- Noise energy E_{noise} ~ kT/clock cycle
- SNR is ~ CV^2 / kT
- $p_{error} = \exp(-E_{signal} / \text{kT}) = \exp(-CV^2 / \text{kT})$

Circuit:



Timing:
Assume clock rate ≈ 1/*f*



(Oversimplification, every node will have a different *C* and fanout. Does not change conclusion.)

Backup: Alternate explanation using channel capacity

- Bandwidth-independent channel capacity

$C = B \log_2(1 + S / (BkT)) = S / (kT) \log_2(1 + S / (BkT)^{BkT/S}$,

where C is channel capacity (bits/second); B is bandwidth; S is signal power; k is Boltzmann's constant; T is temperature. Note $\lim_{x \to 0} (1+x)^{1/x} = e$.

$C_\infty = lim_{B \to \infty} C = (\log_2 e)\ S / (kT)$

In our terms, in a clock period (or any arbitrary time) $N_{bits} = (\log_2 e)\ E_{signal} / (kT)$

However $N_{bits} = \log_2(1/p) = \ln(1/p) / \ln 2$, where p is the probability of a 0 or 1. Rearrange to $\ln(1/p) / \ln(2) = N_{bits} = E_{signal} / (kT \ln 2)$ and exponentiate to get

$p = \exp(-E_{signal} / kT)$

Which is p_{error} if we were expecting a specific value

(Ref: http://www.dip.ee.uct.ac.za/~nicolls/lectures/eee482f/04_chancap.pdf).

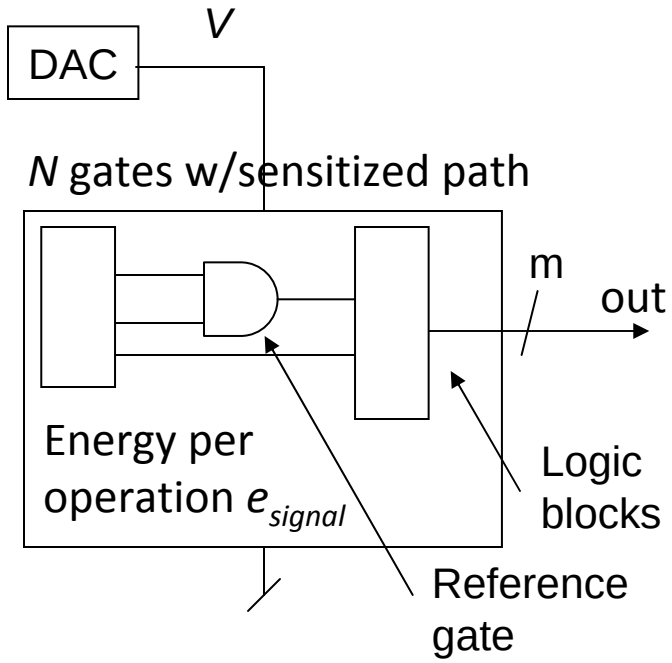
Computer model to exploit energy-reliability tradeoff

- If industry succeeds in finding the "millivolt switch"…
- We should be able to power a chip at changeable voltage *V*
- This will create $e_{signal} = CV^2$
  - With *C* potentially different for each node (wire)
- This will lead to circuits where

$$p_{error} = N \exp(-e_{signal} / kT)$$

- However, the previous equation is oversimplified. Every node or net in a chip will have a slightly different energy efficiency. So say

$$p_{error} = \alpha N \exp(-\beta e_{signal})$$



- *N* is the number of gates with a sensitized path to an output of the circuit; errors in other gates are said to be "masked."

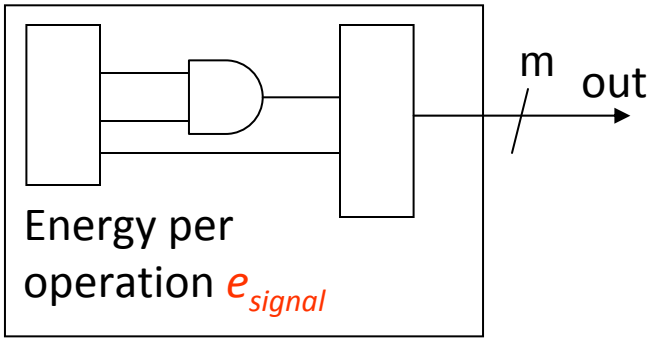
Outline

- Energy-reliability tradeoff
- Error correction for logic
- Redundant Residue Number System
- “Creepy” architecture
- Programming (assertion language)
- Integration with Processor-In-Memory-and-Storage (PIMS)
- PIMS programming

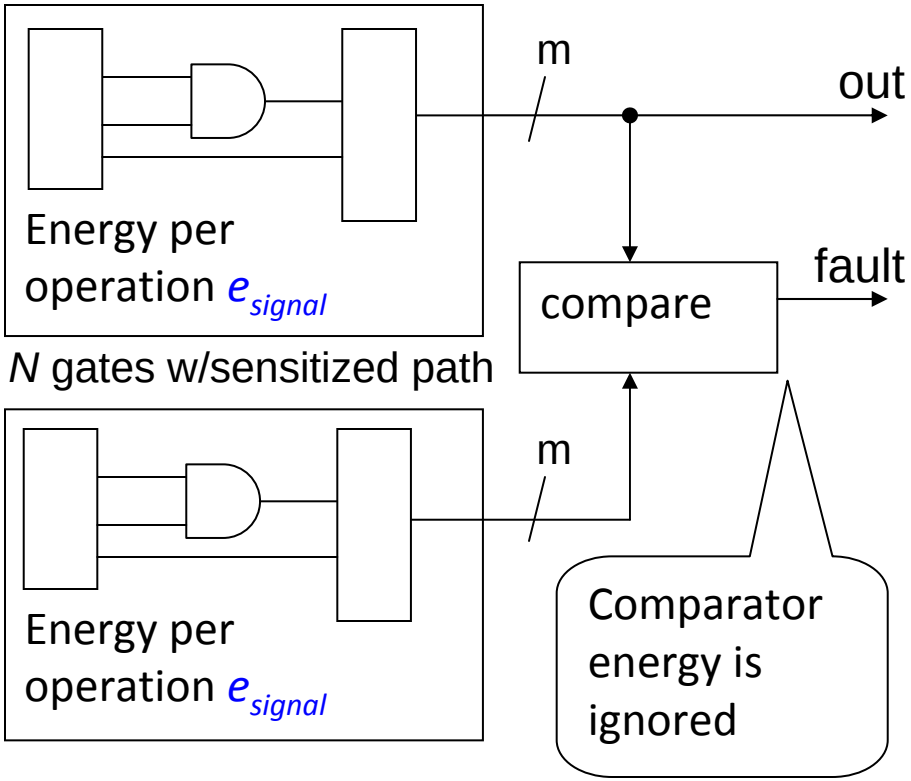
Limits of logic error correction

- Which is better, left or right?
  - Same energy if $e_{signal} = 2\ e_{signal}$
  - Same probability of undetected fault
  - However, right detects many non-energy fault classes

N gates w/sensitized path



N gates w/sensitized path



$$p_{error} = N \exp(-e_{signal} / kT) \quad \underset{\text{(could be equal)}}{\leq} \quad p_{undetected} = \frac{N^2 \exp(-e_{signal} / kT)^2}{N}$$

The "≤" is from errors that create the same output syndrome. Does not affect the exponential.

Channel capacity equivalent for logic?

- Shannon's channel capacity theorem says error correction can be beneficial, but you cannot use it to beat channel capacity limits
- Previous VG is an example of using error detection (correction) for logic...
  - However, it didn't work; we tried to beat the limit and failed
  - Energy is the equivalent to channel capacity
  - Computing something with a fixed probability of error is equivalent to information
- This all depends on $p_{error} = \exp(-Energy / kT)$
  - A different functional relationship would yield a different result

Outline

- Energy-reliability tradeoff
- Error correction for logic
- Redundant Residue Number System
- “Creepy” architecture
- Programming (assertion language)
- Integration with Processor-In-Memory-and-Storage (PIMS)
- PIMS programming

Primer on Redundant Residue Number System

Residue Number System (RNS)

- Given a set of relatively prime moduli m_1, m_2, m_3, m_4, e. g.
  - 199, 233, 194, 239
- Any number < $m_1 \times m_2 \times m_3 \times m_4$ can be represented by the four remainders (residues) upon division by m_j
- Addition and multiplication become vector-wise modular add and multiply
- Comparison, shifting, conversion are *residue interacting functions*

- Redundant RNS (RRNS)
- Add extra moduli, m_5, m_6, e. g.
  - 251, 509
- Up to two bad residues can be detected
- Up to one bad residue can be corrected

- NOTE: Covers the math, not just the storage!

This is the RNS used in Watson, Richard W., and Charles W. Hastings. "Self-checked computation using residue arithmetic." *Proceedings of the IEEE* 54.12 (1966): 1920-1931.

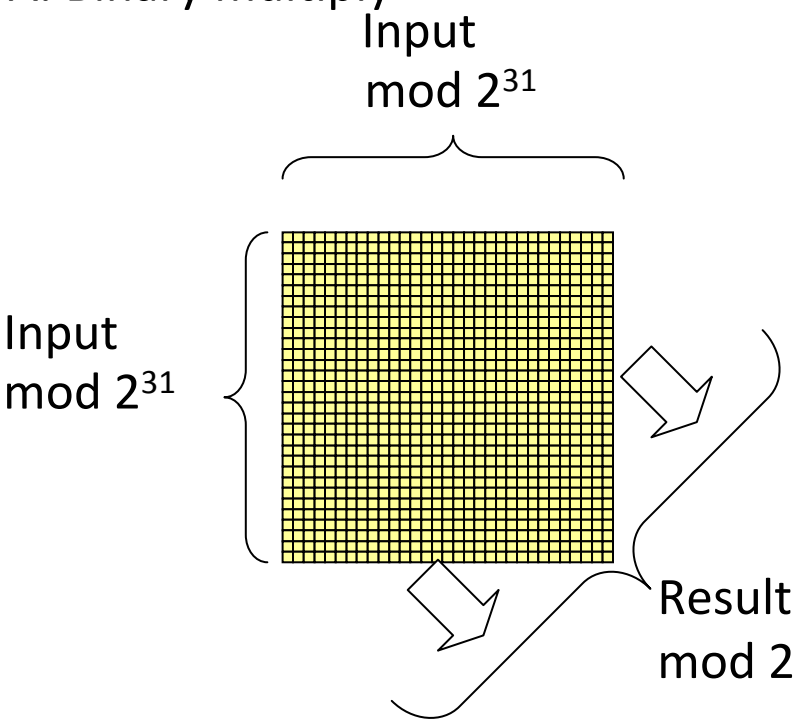
Outline

- Energy-reliability tradeoff
- Error correction for logic
- Redundant Residue Number System
- “Creepy” architecture
- Programming (assertion language)
- Integration with Processor-In-Memory-and-Storage (PIMS)
- PIMS programming

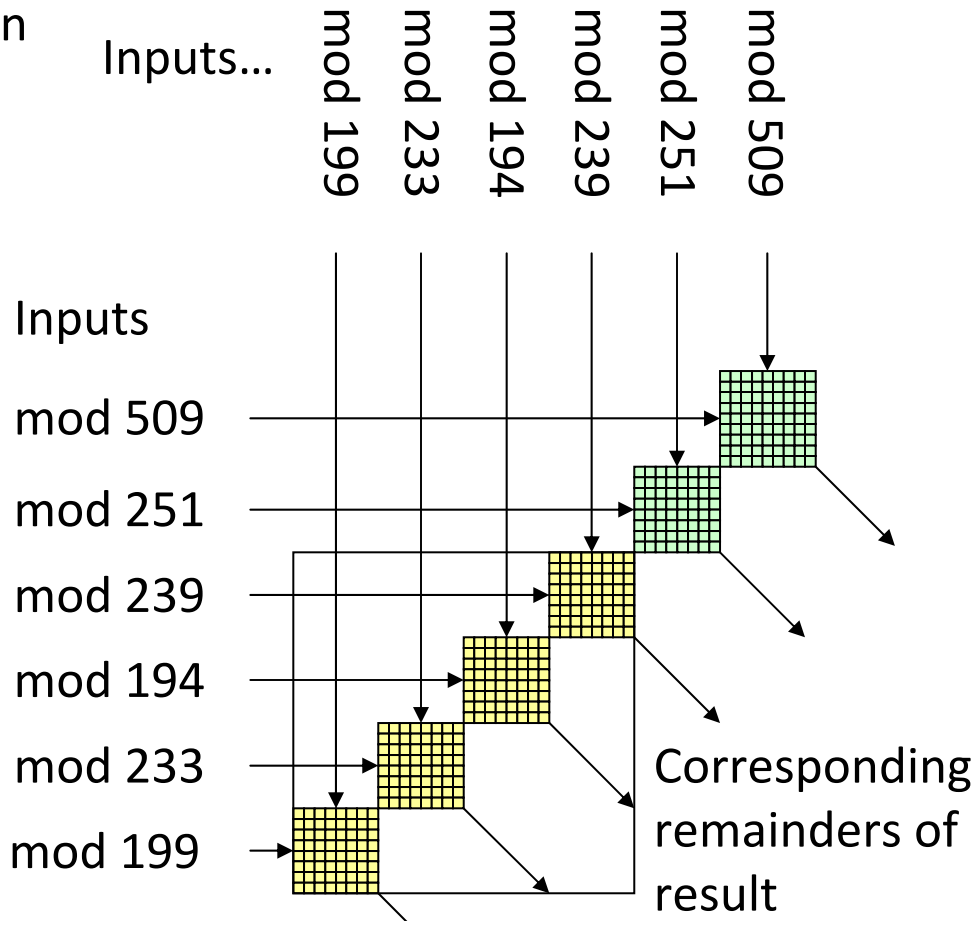
Example where we gain energy efficiency

- Added energy for redundancy in part B is about 50%, so energy efficiency improves given baseline on earlier VG.

A. Binary multiply

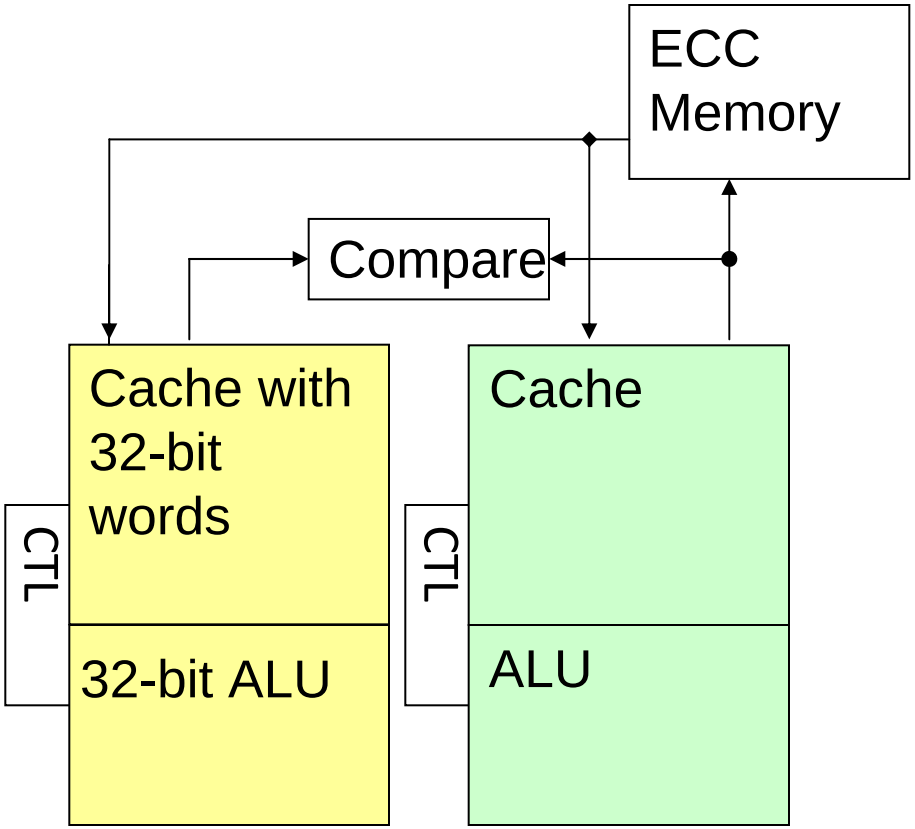


B. Redundant Residue Number System



This is the RNS used in Watson, Richard W., and Charles W. Hastings. "Self-checked computation using residue arithmetic." *Proceedings of the IEEE* 54.12 (1966): 1920-1931.

Baseline: Dual core approach (won't help energy)

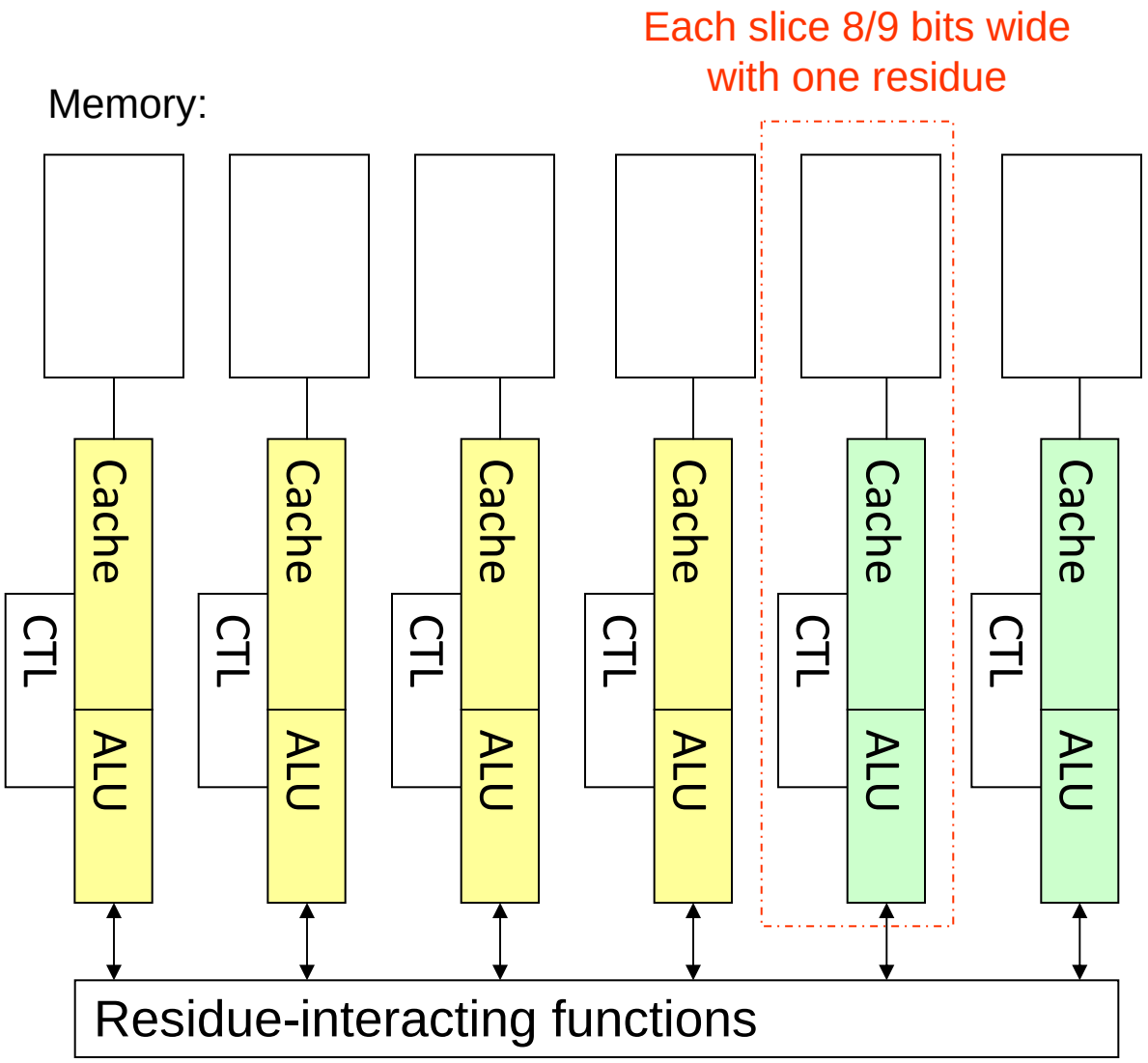


Detection: Memory access different

Correction: Rollback (idempotency argument)

Overhead: 100% on core

Creepy architecture (temporary name)



Detection: Number system inconsistency

Correction: The state of any node can be reconstructed from the others

Overhead: 50% on ALU and cache; 6× on control

Yellow is original data; green is check

Creepy architecture instruction set

Sandia National Laboratories

- Slices run in lock step (except during error processing)
- Slices have two types of state
  - Local state, like loop indices
  - “Residue state,” where each slice uses a different modulus
- Slices may interact with “residue interacting functions”
  - Comparing residue numbers
  - Consistency check a. k. a. error detection
  - Conversion, division, square root, etc.
- Conditional branch
  - Allowed on local state
  - Prohibited on “residue state,” but
  - Allowed on result of residue interaction functions

Creepy architecture error recovery

- Slices run in lock step (except during error processing)
- Error detection
  - If residue consistency check fails
  - Residue consistency check deemed failed if a processor crashes for any reason and fails to participate in a consistency check
- Error recovery
  - On error, five processors will be good and one bad
  - The five good ones regenerate all state for the failed one
  - Restart the failed one

Outline

- Energy-reliability tradeoff
- Error correction for logic
- Redundant Residue Number System
- “Creepy” architecture
- Programming (assertion language)
- Integration with Processor-In-Memory-and-Storage (PIMS)
- PIMS programming

Programming with assertion language

RRNS structure definition with assertions (*ED*=error detect; *EC*=error correct):

```
struct RRN { int r199:8, r233:8, r194:8, r239:8, r251:8, r509:9; }
    assert(ED(...)) error(EC(x,...));
```

Multiply:

```
struct RRN mul (RRN a, RRN b) { v, p_u(...), p_d(...), E(...) } {
    return RRN (a.r199*b.r199%199,
        a.r233*b.r233%233,
        a.r194*b.r194%194,
        a.r239*b.r239%239,
        a.r251*b.r251%251,
        a.r509*b.r509%509);
}
```

p_u(...), p_d(...), E(...) are pragmas conveying information on error probabilities and energy consumption to the system

Hans P. Zima, Erik DeBenedictis, Jacqueline Chame, Pedro C. Diniz, Robert F. Lucas, *The FailSafe Assertion Language Version 8.0*, Technical Report, Information Sciences Institute, University of Southern California, May 2015

Managing the energy-reliability tradeoff

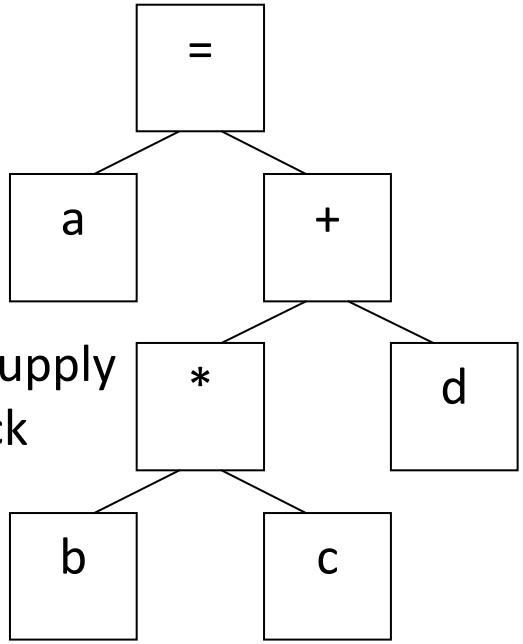
Example:

```
a = b*c+d
```

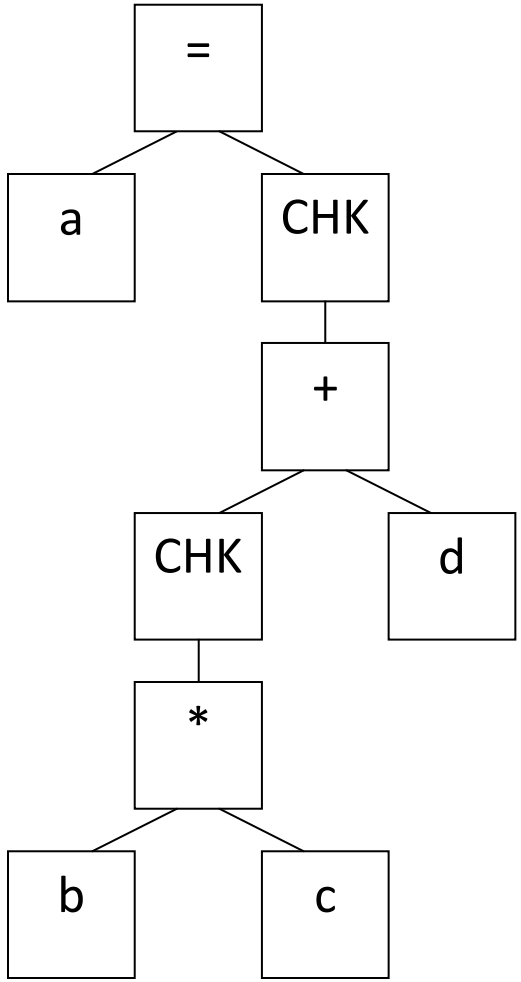
Compiler and run time system may:

* Evaluate assertions or not do so
* Raise or lower power supply
* Use checkpoint/rollback
* Propagate properties up the hierarchy, so only bottom levels need to be annotated

A. High *v*, low error rate:



B. Low *v*, high error rate:



Outline

- Energy-reliability tradeoff
- Error correction for logic
- Redundant Residue Number System
- “Creepy” architecture
- Programming (assertion language)
- Integration with Processor-In-Memory-and-Storage (PIMS)
- PIMS programming

Overview

- Let's make an extremely energy efficient system using different methods
- Memory: High power but recycled
  - Adiabatic
- Processor: Low power but error correction
  - Error-corrected computing

Optimal Adiabatic Scaling

- From a different talk
- This is an adaptation of a scaling rule to memory
- Based on adiabatic scaling where energy $\propto f^2$
- Let's plot economic quality of a gate or chip:

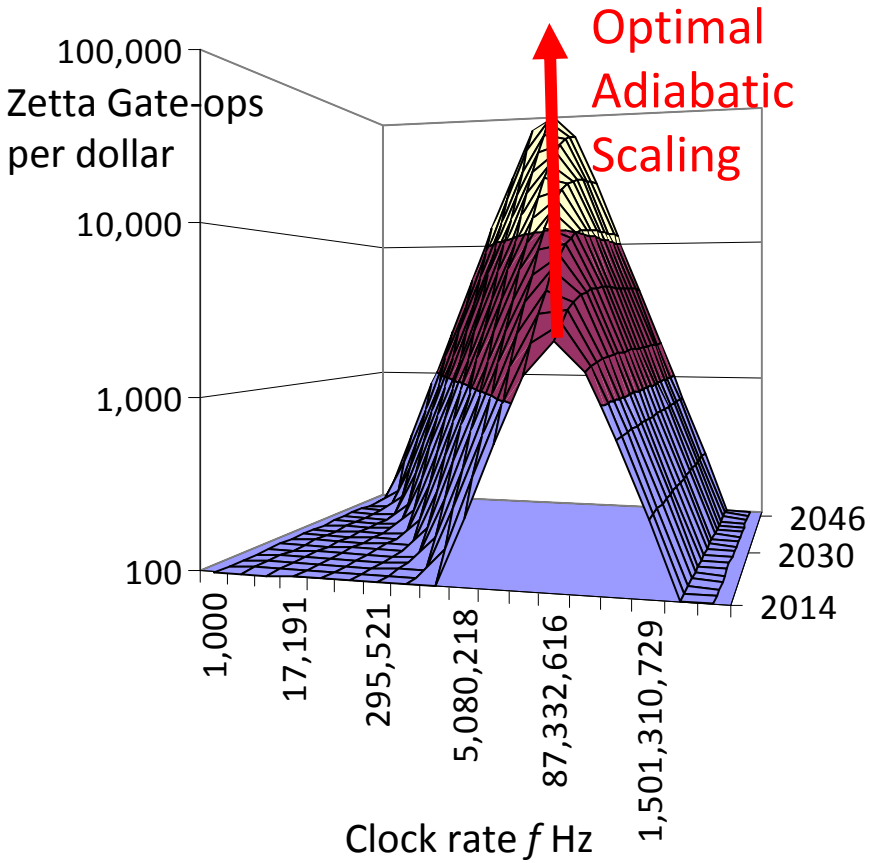
$$Q_{chip} = \frac{Ops_{lifetime}(f)}{\$_{purchase} + \$_{energy}(f^2)}$$

Where $\$_{purchase} = A\ 2^{-t_{year}/3}$

$Ops_{lifetime} = Bf$, *and*

$\$_{energy} = Cf^2$ (A, B, and C constants)

- Assume manufacturing costs drops to ½ every three years
- Top of the ridge rises with time



Resulting scaling scenario (standard chart with additional column)

If *C* and *V* stop scaling, throughput ($f\ N_{tran}\ N_{core}$) stops scaling.

	Const field	Constant *V*				Optimal Adiabatic Scaling
		Max *f*	Const *f*	Const *f*, N_{tran}	Multi core	
L_{gate}	$1/\alpha$	$1/\alpha$	$1/\alpha$	$1/\alpha$	$1/\alpha$	1*
W, L_{wire}	$1/\alpha$	$1/\alpha$	$1/\alpha$	1	$1/\alpha$	$N=\alpha^2$†
V	$1/\alpha$	1	1	1	1	1
C	$1/\alpha$	$1/\alpha$	$1/\alpha$	1	$1/\alpha$	1
$U_{stor} = ½\ CV^2$	$1/\alpha^3$	$1/\alpha$	$1/\alpha$	1	$1/\alpha$	$1/\sqrt{N}=1/\alpha$‡
f	α	α	1	1	1	$1/\sqrt{N}=1/\alpha$
N_{tran}/core	α^2	α^2	α^2	1	1	1
N_{core}/A	1	1	1	1	α	$\sqrt{N}=\alpha$
P_{ckt}	$1/\alpha^2$	1	$1/\alpha$	1	$1/\alpha$	$1/\sqrt{N}=1/\alpha$
P/A	1	α^2	α	1	1	1§
$f\ N_{tran}\ N_{core}$	α^3	α^3	α^2	1	α	$\sqrt{N}=\alpha$

Theis and Solomon → New

Under optimal adiabatic scaling, throughput continues to scale even with fixed *V* and *C*

* Term redefined to be line width scaling; 1 means no line width scaling

† Term redefined to be the increase in number of layers; previously was 1 for no scaling

‡ Term redefined to be heat produced per step. Adiabatic technologies do not reduce signal energy, but "recycle" signal energy so the amount turned into heat scales down

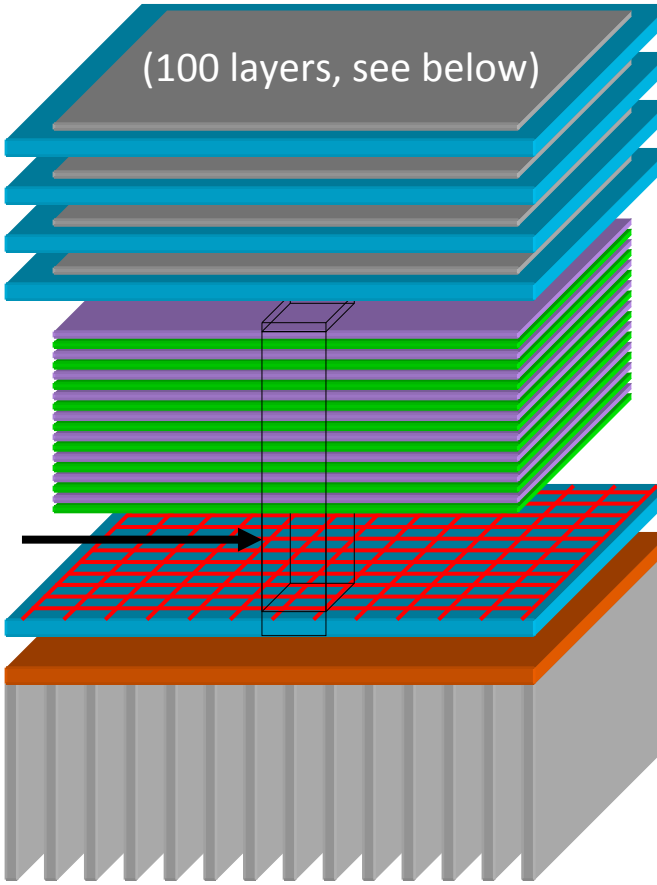
§ Term clarified to be power per unit area including all devices stacked in 3D

Ref: T. Theis, In Quest of the "Next Switch": Prospects for Greatly Reduced Power Dissipation in a Successor to the Silicon Field-Effect Transistor, Proceedings of the IEEE, Volume 98, Issue 12, 2010

Processor-In-Memory-and-Storage (PIMS) Physical implementation vision

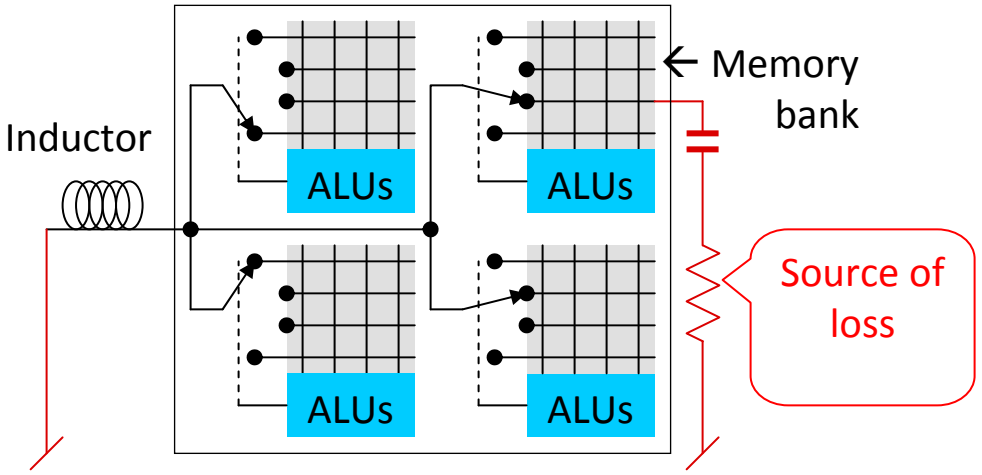
From a different project

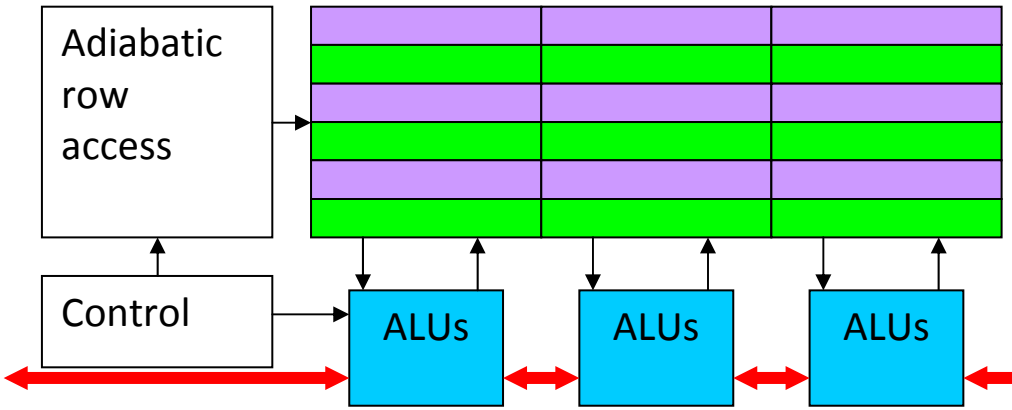
- Storage/Memory
  - Flash, ReRAM (memristor), STM, DRAM
- Base layer
  - PIMS logic
- 3D
  - Whole structure is layered



PIMS architecture

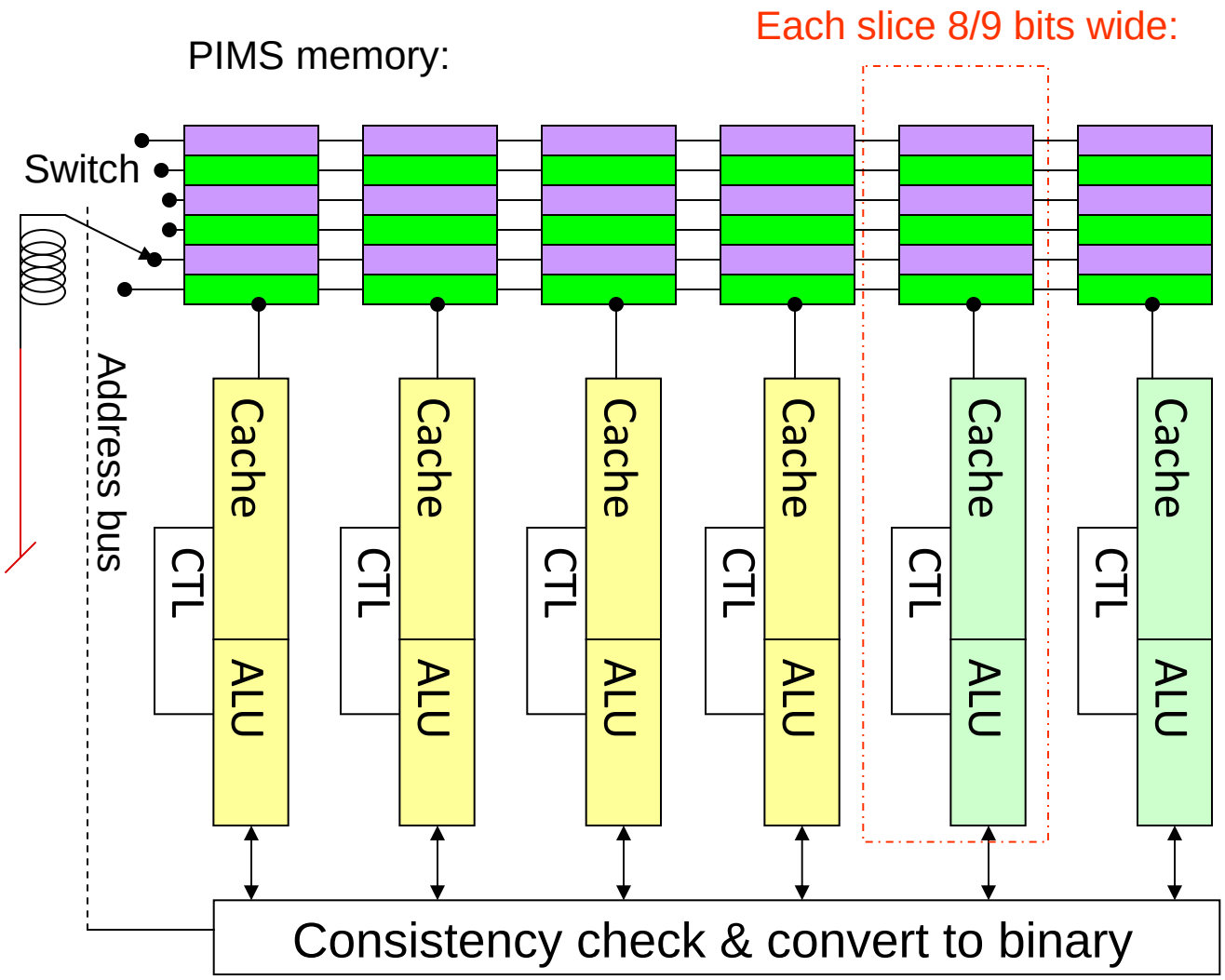
- Adiabatic memory structure →
  - Adiabatic memory banks, all clocked the same but different rows
- PIM processors
  - Each row has multiple ALUs
- Programming
  - Memory has opcodes and data controlling arithmetic and communications on ALUs





PIMS memory/storage:
Purple is the configuration "opcode"
Green is memory contents
Red is communications between ALUs

RRNS architecture connected to PIMS (Creepy architecture)



RRNS used for both data and memory addresses (novel?)

Adiabatic memory; OAS scaling rule

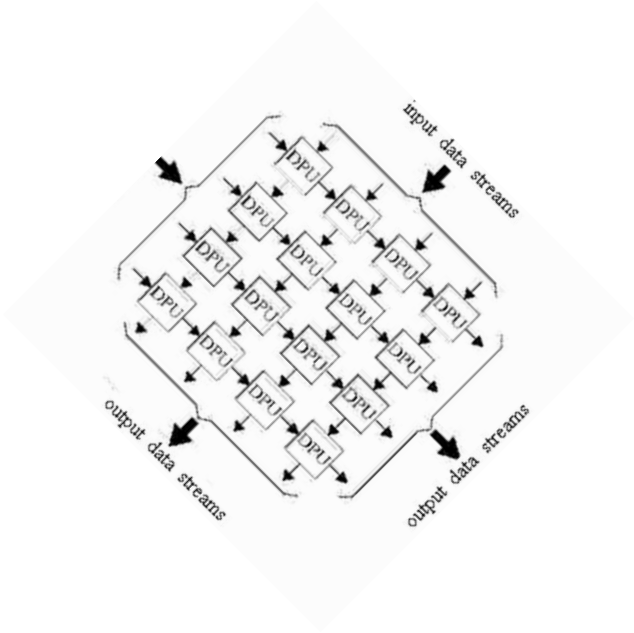
Logic at point of optimal energy-reliability tradeoff

Runs legacy code – given augmentation with assertion language

Outline

- Energy-reliability tradeoff
- Error correction for logic
- Redundant Residue Number System
- “Creepy” architecture
- Programming (assertion language)
- Integration with Processor-In-Memory-and-Storage (PIMS)
- PIMS programming

Tile programming



x

1	2	3	4

A

1	0	0	2
0	0	3	0
0	4	0	5
6	0	0	0

=

y

25	12	6	17

Vector-matrix multiply on left implemented by dataflow-like spreadsheet below.

Timestep 1:

Timestep 2:

Etc.

			x_0 1		y_0 0	
		x_1 2		a00 1 x_0 1 y_0 1		y_1 0
	x_2 3		a10 0 x_1 2 y_0 1		a01 0 x_0 1 y_1 0		y_2 0
x_3 4		a20 0 x_2 3 y_0 1		a11 0 x_1 2 y_1 0		a02 0 x_0 1 y_2 0		y_3 0
	a30 6 x_3 4 y_0 25		a21 4 x_2 3 y_1 12		a12 3 x_1 2 y_2 6		a03 2 x_0 1 y_3 2	
y_0 25		a31 0 x_3 4 y_1 12		a22 0 x_2 3 y_2 6		a13 0 x_1 2 y_3 2		
1st cell column above, as it evolves with time	y_1 12		a32 0 x_3 4 y_2 6		a23 5 x_2 3 y_3 17			
	2nd cell column above, as it evolves with time	y_2 6		a33 0 x_3 4 y_3 17				
		3rd cell, and so on	y_3 17					

Note: the y_j's are updated, so they do not all have the same value

Note on above: this diagram is only a spreadsheet, but you may think of a row of x's and y's as a register that shifts right and left each time step; the a's do not shift (see arrows).

Tile programming

x

1	2	3	4

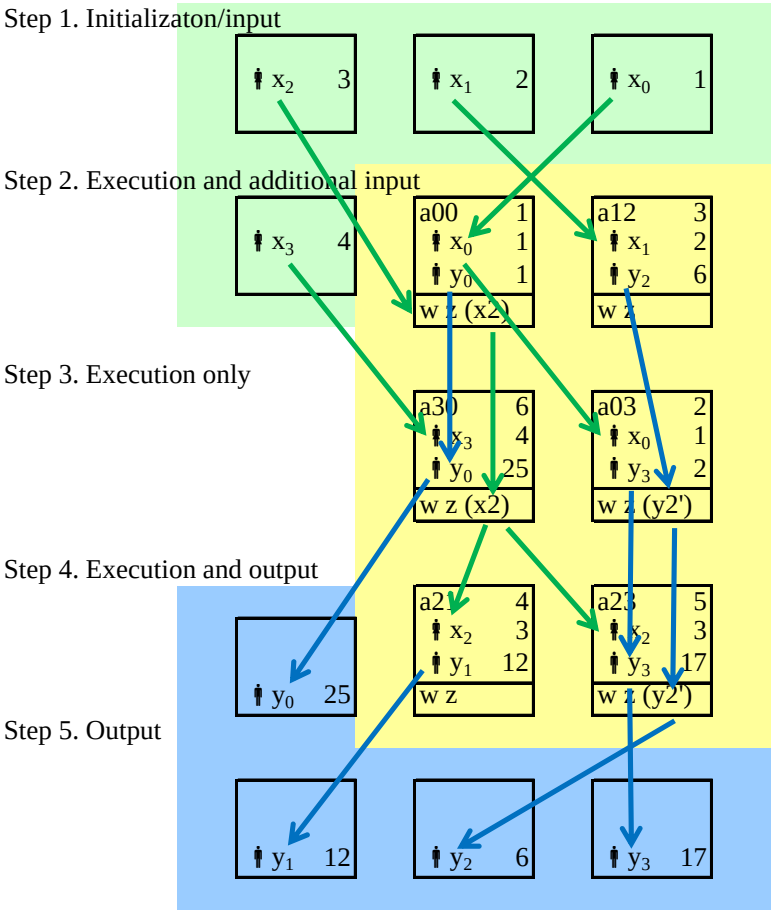
A

1			2
		3	
	4		5
6			

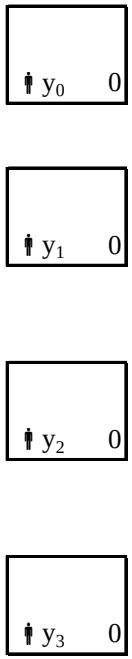
= y

25	12	6	17

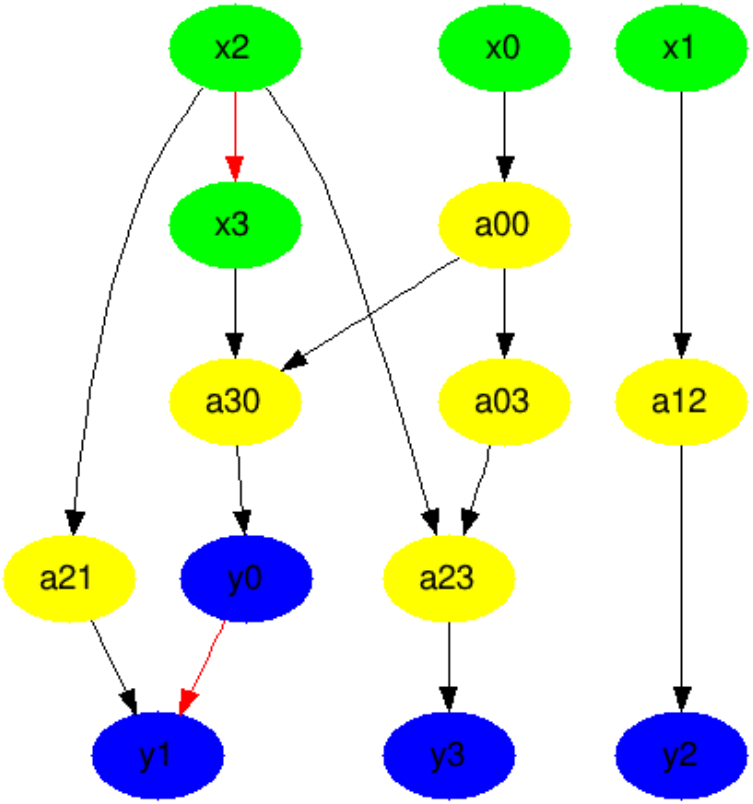
Arrows indicate data flow; wth no data flow faster than nearest neighbor per step. Sometimes dance steps for ladies and gents.



Zeros



GraphViz:



Conclusions I

- Semiconductors heading to a non-abrupt limit
- The limit comprises a particular functional relationship between energy and errors
- Can we develop an error detection and correction approach for this relationship?
- If we succeed
  - Estimate 2:1 energy reduction (one generation)
  - Will fix errors due to Cosmic Rays, weak parts, etc. so we may have to do it anyway

Conclusions II

- What is the impact of a 2:1 boost in energy efficiency?
- In the past, this level of boost would get run over by Moore's Law in 18 months. If it delayed your progress by more than 18 months, it would be a loser
- However, it this moves out the endpoint of Moore's Law by 2:1, the benefit will accumulate forever

Conclusions III

- The assertion language may be needed sooner
- Exascale documents list reliability in the top several risk factors
- If we have to address reliability in a systematic way for Exascale, we might as well do it in a way that could be used for power efficiency later

Challenging issues

- To save energy, logic error detection requires very light-weight circuits. More research will be needed on the circuits.
- In a computer, error detection and correction can be done as an exception or interrupt. However, this time becomes overhead that reduces the amount of useful work done
- Other issues could interact with the topics in this presentation
  - While thermal errors are unavoidable, other error classes could get in the way first. Examples: Cosmic Rays, weak parts, aging, device-to-device variance
  - The methods here could support reliable overclocking, essentially recognizing an “excessive overclocking error.” This generalizes to using error detection to convert design safety margin to performance
- If this approach catches on, hardware design tools and software may need enhancement

Abstract

It is well known that the rise in computer energy efficiency due to Moore's Law is associated with a rise in error rates, but we show how to turn this effect into a benefit. Error rates do not rise abruptly, but will grow exponentially as energy is reduced. With the energy vs. error rate tradeoff quantified, it is possible to derive management strategies.

Computing with low-energy but unreliable components has been explored elsewhere as approximate computing (where some errors pass through to the user) and random computing (where errors become random numbers used in algorithms). However, we devise a third approach where error correction makes computation reliable without the error correction itself consuming more energy than gained.

We will discuss a fault-tolerant architecture tentatively named "creepy," which uses residue arithmetic to correct errors at high error rates while still being energy efficient.

By properly managing errors with the new architecture, it may become possible to utilize a final few generations of semiconductor scaling beyond what can be exploited by the traditional microprocessor.