Approximate Computing [1]: Trading Accuracy for Efficiency

Alex Anisimov, Steven Johnson

10/31/2025

Connection to Dr. Kyle Wilson's Class

The following C code is the fast inverse square root implementation from *Quake III Arena*, stripped of C preprocessor directives, but including the exact original comment text:[15]

```c
float Q_rsqrt( float number )
{
	long i;
	float x2, y;
	const float threehalfs = 1.5F;

	x2 = number * 0.5F;
	y  = number;
	i  = * ( long * ) &y;                       // evil floating point bit level hacking
	i  = 0x5f3759df - ( i >> 1 );               // what the fuck?
	y  = * ( float * ) &i;
	y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
//	y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed

	return y;
}
```

[2, 3]

The Computing Challenge

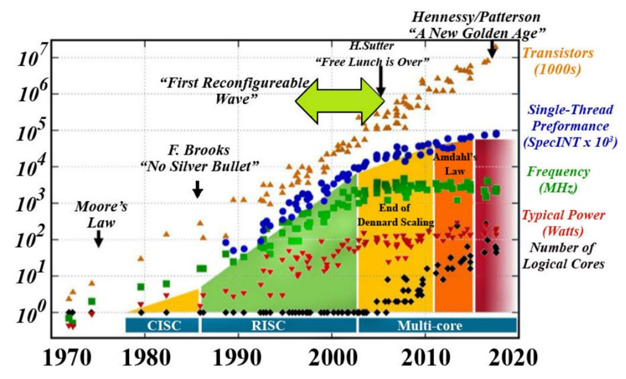


FIGURE 1. 42 years of microprocessor trend data [6], [7].

- Exponential data growth
- Increasingly complex computations
- Slowing hardware efficiency improvements
- Environmental concerns
- Energy consumption

Key idea

Many applications can handle a small error. Therefore, let's develop algorithms producing results close to the targets.

Reduced energy consumption

Up to 42%

Faster execution

Up to 27%

Smaller hardware requirements

Where Can We Use Approximate Computing?

✔	✘
• Multimedia processing (image/video/audio) • Machine learning and AI • IoT sensor data processing • Data mining and analytics	• Medical systems • Military systems • Financial transactions • Other safety-critical systems

The Four-Level Framework

1. **Data Level** — Approximate the input data
2. **Software Level** — Optimize code and algorithms
3. **Architecture Level** — Modify memory and processors
4. **Circuit Level** — Redesign arithmetic units

This levels can be combined for greater efficiency

Overall Framework

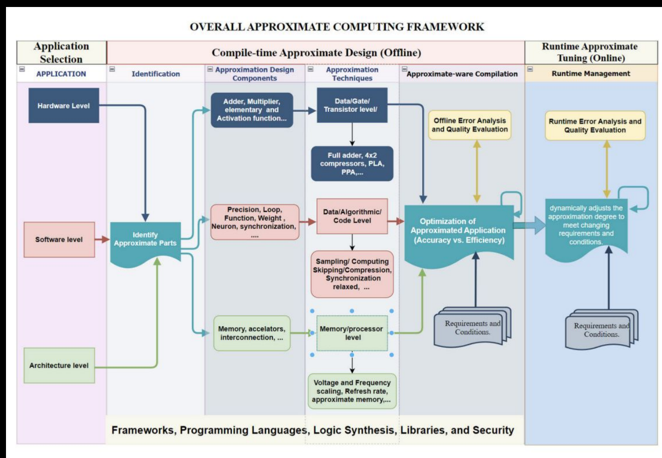
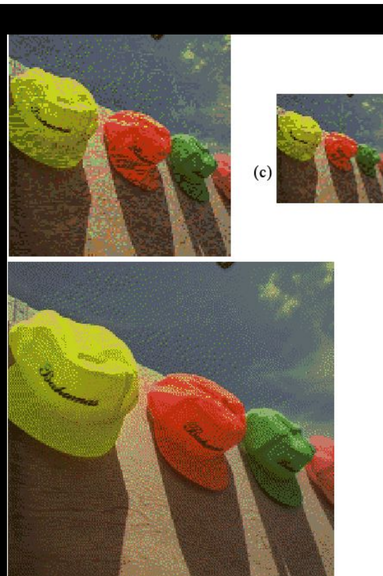


FIGURE 5. Overall framework of approximate computing.

Three Integrals Parts:

- Selection of Error-Tolerant Applications
- Offline AxC
- Online AxC



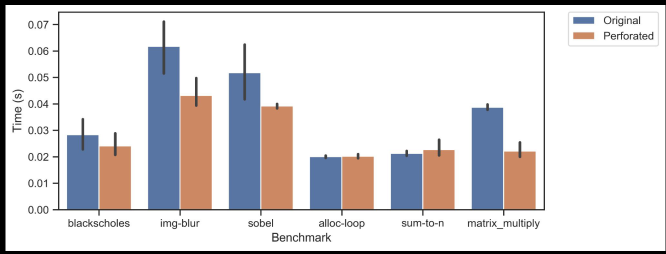
Color Quantization Results

Data-Level Approximations Methods

- Data sampling (process subset of data)
- Quantization (reduce precision)
- Compression
- Probabilistic data structures (e.g., Bloom Filters, HyperLogLog, MinHash, T-Digest)

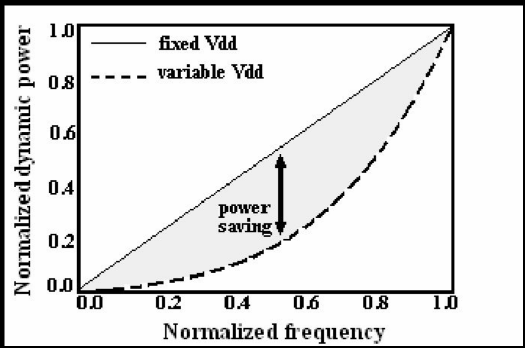
Software-Level Approximations Methods

- Code optimization
  - Loop perforation (skip iterations)
  - Early stopping
  - Pruning (remove unnecessary computations)
  - Function approximation
  - Approximate memoization
- Approximate parallelism and relaxed synchronization
- Specialized frameworks and languages for AxC



Results of loop perforation

Architecture-Level Approximations Methods



Dynamic voltage and frequency scaling power savings

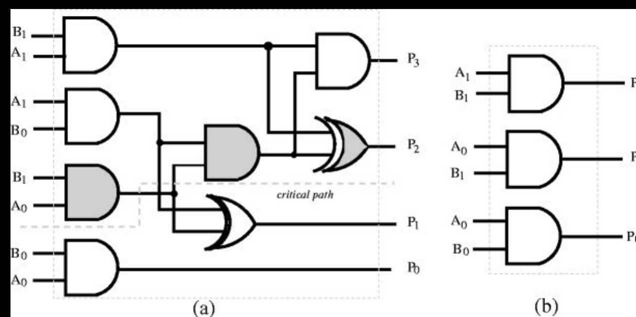
- Approximate memory
  - Reduced refresh rates
  - Voltage scaling
  - Process-in-memory (PIM)
- Voltage/Frequency management
  - Dynamic voltage and frequency scaling (DVFS)
  - Near-threshold voltage operations
- Approximate processors (co-designed software/hardware units)

Circuit-Level Approximations Methods

Focus on arithmetic units.

Examples:

- Approximate adders
- Approximate multipliers
- Approximate dividers



a) Accurate multiplier; b) Approximate 2-bit multiplier

Why Is It Relevant?

AI's computational demands:

- Billions of parameters when training models
- Millions of requests per second

How can approximate computing help?

- Neural networks are inherently error-tolerant
- Small weight/activation changes don't significantly affect accuracy
- Training uses noisy gradient descent anyway
- Many AI applications (computer vision, NLP) tolerate some imprecision

Quantized MAC with Shift-and-Add

Problem: every neuron performs Σ(weight × input) + bias

- Truncate least significant bit from both inputs and weights
  - In NN's, precision in LSB contributes the least to the final answer
- Right-shift instead of multiply
- Weight elimination
  - Skips computations entirely if weight is zero
- Iterative accumulation and status check
  - Stop after fixed number of iterations or when contribution becomes negligible

42%
Resource reduction

27%
Delay reduction

Emerging Technologies and Future Directions

1. **Cross-Layer and End-to-End AxC**
   Integrate all four layers of approximate computing
2. **Shannon-inspired statistical computing**
   Leverage statistical properties of data and hardware
3. **Brain-inspired computing**
   Work on neuromorphic circuits mimicking the brain
4. **Automated AxC selection**
   Experiment with AI identifying when and how to approximate
5. **Domain Specific Frameworks**
   Develop tailored AxC tools for different solutions
6. **Error bound guarantees**
   Work on formal verification methods

Significance

✔

- Provides actionable framework and analyzes ways to achieve results
- Demonstrates practical impact (up to 42% gains)
- Essential for substantial AI growth

✘

- Context-dependent availability
- Requires careful error analysis and quality assurance
- Not applicable in critical industries

Thank you.

Any questions?

Let's talk

Potential discussion topics

Where else could approximate computing be applied?
What are ethical implications of good enough computing?
How do we determine error bounds?

References

[1] A. M. Dalloo, A. J. Humaidi, A. K. Al Mhdawi, and H. Al-Raweshidy, "Approximate computing: Concepts, architectures, challenges, applications, and future directions," *IEEE Access*, vol. 12, pp. 118345–118372, Sept. 2024, doi: 10.1109/ACCESS.2024.3467375.

[2] id Software, “q_math.c,” *Quake III Arena — code/game/q_math.c*, GitHub repository (archived), July 29 2017. [Online]. Available: https://web.archive.org/web/20170729072505/https://github.com/id-Software/Quake-III-Arena/blob/master/code/game/q_math.c#L552 [Accessed: Jan. 21, 2017].

[3] “Fast inverse square root,” *Wikipedia, The Free Encyclopedia*, last edited 29 October 2025. [Online]. Available: https://en.wikipedia.org/wiki/Fast_inverse_square_root. [Accessed: Oct. 30 2025].