**What is Approximate Computing**

* To achieve a satisfactory computational result with reliable and controllable error thresholds, in other words: relaxed precision. Basically, as Jie Han form the University of Alberta puts it, "...[it] employs active design methodologies that exploit the nature that many systems and applications can tolerate some loss of accuracy in the computation result." These kinds of algorithms are computationally faster algorithms that use less energy, yielding inexpensive results of lower quality with negligible error
* Levels of Approximate Computing
  + Logic & Circuit
  + Microarchitecture (atomic functions)
  + Algorithms
  + Parameters
* Is undeniably faster and lower-power as it is unreliable
* Classes of Approximate Computing Techniques
  + Error Occurrence
    - Either non-deterministic (errors happen randomly) or deterministic (achieve same every time)
  + Degradation
    - Concerned with error rates and outcomes (magnitude and frequency of error tolerance )
    - Error rates refer to toggle-able (switch between approximate and precise operation) or gradual degradation (relaxing precision or error checking mechanisms over time as certain conditions are met )
    - Error outcomes refer to
      * Bounded (Toggleable, approximation may be switched off if errors exceed error bound(s))
      * Catastrophic (Togglable and Gradual, severe failures that occur from exceeding error bound over time; relaxed )
      * Graceful (Gradual)
  + Level
    - Transistor, logic, or algorithmic
  + Evaluation
    - Atomic (evaluating error introduced in atomic operations e.g. a filter applied to a single pixel) or Application (evaluate execution time, energy consumption, and output quality)
* The success of an approximated application is its reliability
* Reliability looks like executing the process multiple times, and the end-state of the relaxed version matches the original version a specified number of times, we deem the program reliable to that point

**Practical Application**

* Code Perforation (Skipping Iterations in a Loop), Function Substitution, Approximate Memoiztation (Caching Results from Expensive Function Calls), Relaxed Synchronization (Lock Elision), Approximate Hardware
* For example, choosing between using single and half precision numbers is a difference of ~3 bytes/13 bits
* In unreliable hardware such as: an ALU that concludes 2+2 = 6 or Secondary Memory that interprets the memory location 0x3452 (12 in decimal) but retrieves from decimal memory location 40
* Voltage Scaling + Dynamic Voltage and Frequency Scaling (DVFS).
  + This means reducing the operating voltage of the processor. The technique even goes as far as to adjust the voltage dynamically based on the input/workload.
* Error Resilient Algorithms
  + Algorithms such as the Newton-Raphson method, lossy compression (like JPEG), sorting algorithms like TeraSort (which highly performant but slightly imprecise),
* Application Approximation
  + This means instead of approximating the entirety of an application, we approximate only parts of it while leaving other parts of it with higher levels of accuracy
* Quality-Aware Programming
  + A programming tactic which requires that software developers be keenly aware and constantly attempting to reduce the space complexity of a program as much as possible.

**Real Life Examples**

* Often used to meet project budget or time requirements
* Well known approximation methods include fixed point (how decimals are normally represented like 3.2 or 41.23), block floating point (used in digital signal processing), and floating point
* Apporximate Computing helps in computing clusters used for deep learning or LLMs. The amount of heat generated from those clusters is enormous and requires a ton of power to keep cool. Reducing the amount of energy needed to power the clusters and cool them would mean money saved, cheaper and smaller hardware, and more computational efficiency per unit of space in such a facility.
* "...approximate computing techniques need to exploit the error-tolerance of humans and neural networks. This optimization can lead to lightweight neural networks." (21, Srivastava, Srishti, et al. "A Survey of Deep Learning Techniques for Vehicle Detection from UAV Images." CSE Department, IIT Dharwad, India, ECE Department, NIT Trichy, India, ECE Department, IIT Roorkee, India, 2024.)
* Stochiastic/Probabilistic Computing, which is designed to infer based on data (e.g. a person buys yoga pants, a set of dumbbells, books on nutrition: make an inference). Requires testing many scenarios and noting features that are freqently present in all scenarios
* Video Compression
* Image Compression
* Max Pooling
* ANSYS Mesh Density
* Function Approximation p
* Single vs. Double Precision GPU computation with Roofline Graph

**Under The Hood**

* General Steps
  + Achieve minimal error result
  + Establish safe execution envelope
  + Relax semantics of program
  + Verify program result
* Uses assertions (e.g. a / b such that b ≠ 0, where b≠0 is the assertion)
* If the relaxed version of a program is different to the original, when will they converge? When can they be related again?
  + There are synchronization points where the 2 converge, and often times those
  + We bring the assertion or result to a previous (or sometimes future) synchronization point and verify it there
  + Verification reuses existing reasoning from the original program
  + Languages such as Coq analyze raw source code using user-given inference rules to validate logic
* Prominent concerns with radically inaccurate results, rarely correct results, crashes or other malicious actions
* These concerns are avoided via error bounding
* First thing’s first: develop quantitative verification systems
  + E.g. functions that verify the reliability of values or operations
* Error Resilience Domains are application environments where approximate results are acceptable
  + Human Perception (e.g. things we can’t hear)
  + Data Redundancy (e.g. noisy data)
  + Generally, areas that have “No Golden Result”
* End performance can be measured via things like data processing and feature extraction
* Data processing refers to the quality of filtering, compression, or equivalent action which can be measured by human perception
* Feature extraction refers to the identification of properties or characteristics of a data instance using algorithms/methods
* Quantified Reliability
  + .99 \* R(x,y,src,dest) <= rd(val) \* op(\*.) \* R(val)
  + In English:
    - Specified Bound \* Reliability of Operation <= the probability we read the value right \* the probability the operand performs right \* the probability that the input value itself is right
* Then, we evaluate the reliability of everything (e.g. reading, writing, arithmetic, the representation of values themselves
* It’s important that we use reliable inputs (determined by a function that measures their reliability), because if we put junk in, we get junk out
* Ultimately,
  + Probability operations must stay within the established acceptable error threshold
  + Computations must still meet reliability requirements (though tolerant of errors) which can be ensured using error detection and correction mechanisms or algorithmic adjustments that compensate for potential errors
  + Error-tolerant software must be tested, validated, or verified at an appropriate frequency to ensure reliable results despite the uncertainties introduced by the computer’s hardware or lower-level processes