COMPUTING 10,000x MORE EFFICIENTLY

Joe Bates
MIT Media Lab, Visiting Scientist
Carnegie Mellon CS Department, adjunct prof
Singular Computing llc

web.media.mit.edu/~bates

THE MOTIVATING PROBLEM

- Computations specified by programmers
  are implemented as behavior in physical material

- Hardware designer's job:
  efficiently implement $\underbrace{\textbf{Math}}_{\substack{\text{(near) perfect arith} \\ \text{uniform mem delay}}}$ (what sw wants) using $\underbrace{\textbf{Physics}}_{\substack{\text{noisy, approximate} \\ \text{delay} \sim \text{distance}}}$ (what silicon offers)

- Increasingly difficult as decades passed and transistor counts exploded

- Now each instruction (increment, load register, occasionally multiply)
  invokes >10M transistor operations, even though a single transistor
  can perform, for instance, an approximate exponentiate or logarithm

THE MOTIVATING IDEA

- Suppose we go in the opposite direction,
  move instruction set much closer to physics?
- Programmers will face things usually hidden by CPU design,
  but might gain enormous efficiency (speed, energy, size, cost)
- "Natural Computing"
- Here is how I've tried to do it, and some results....

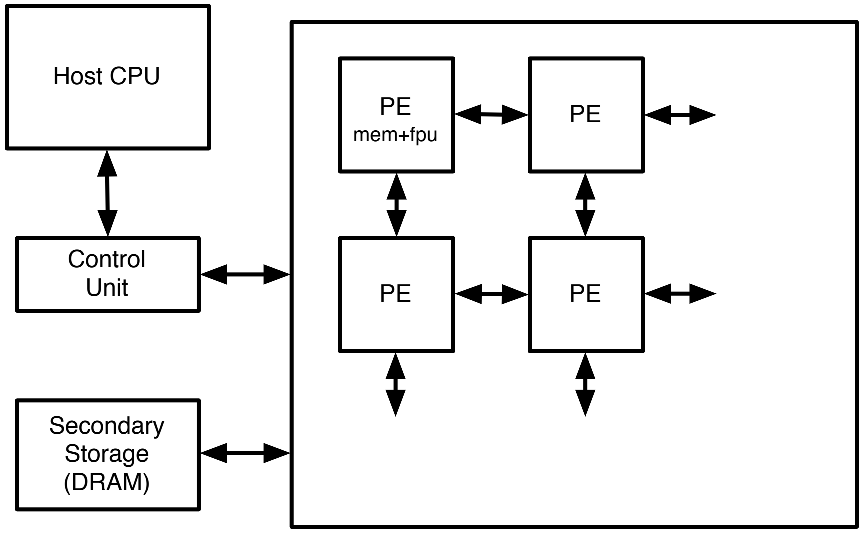
OVERVIEW

- Interested in solving tasks that benefit from floating point ("fp"),
  but IEEE floating point unit takes >500K transistors
- Could less accurate fp arith unit (eg, 1% error) be very small?
- Yes: at least 100x smaller - O(5K) transistors - *will sketch*
- If errors can be compensated in application software,
  can get 10,000x better speed, power than CPU (100x GPU)
- Errors **can** be compensated (in varied apps) - *some examples*
- If hardware cheap and easily available to researchers/students
  could greatly impact computational sciences, CS/AI, medicine, . . .
- This is my overall goal - a research and commercialization effort

ONE PATH TO A SMALL FPU

- Represent values as logarithms
- Choose precision of logs to get 1% precision in numbers
  (6 bit fraction needed, along with perhaps 6 integer bits)
- $\times$ / $\sqrt{\ }$ are small, fast, exact circuits (just add, sub, shift logs)
- $+$ is easy if can compute $F(x) = \log(1+2^x)$ ($-$ similar)
- F can be approximated by small, fast, combinatorial circuit
- Total FPU is ~5K transistors, at ~1GHz

SURROUNDING HARDWARE?



consider classic SIMD co-processor
(MPP, MasPar, DAP, Connection Machine...)

mesh (to obey ~2D physics) (w extensions)

Each PE:
~100 words of 16 bits (float, int, or bits)
math: float +-*/$\sqrt{}$ int +- 16 bit $\wedge \vee \neg$
conditional operations ("masked" PEs)

Advantages of simple SIMD ("single instruction stream, multiple data stream")
- doesn't swamp tiny FPU with other stuff
  => can fit **O(100,000)** PEs on a chip - not 8 or 480 - at O(1GHz)
- so fast, small, power efficient (~Petaop desktop, ~Teraop mobile)
- scales with silicon - doesn't lose it's edge to commodity processors
- well studied in 80s, with known development tools (C*, *Lisp, Fortran, etc)
- easy to understand what's going on (unlike GPU) & easy to build

LIMITATIONS

- SIMD - lockstep parallelism (but with masks)
- Local data flow - distance costs time
- Processing and on-chip bandwidth is Peta,
  off-chip bandwidth is Giga
- Limited memory per PE
- fp arithmetic unusually approximate

What software can run well in this setting?

SOFTWARE EXAMPLES

- Running *(emulation)*
  - Long sums
  - Image kernel operations
  - Tomography
  - Nearest neighbor
- Other plausible tasks

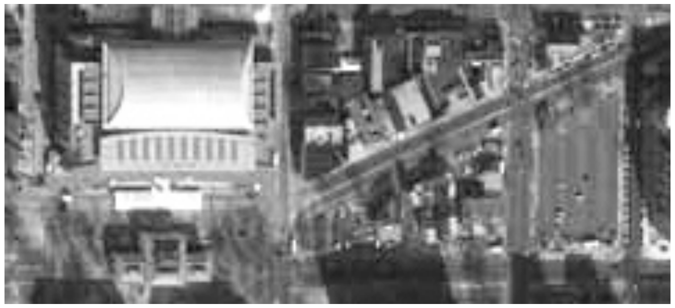
SOFTWARE EXAMPLES

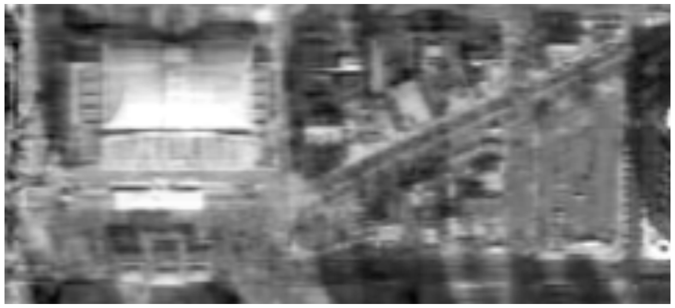
- Basic approach -
  - handle errors in app specific way, not universal hw solution
  - one useful approach - layered like IP stack, each layer reduces error
  - goal: once reach top, reliability sufficient for **that** real world task

- Example: long sums
  - long sums with 1% error may degenerate (not assuming a distribution)
  - Kahan (1965) suggested 4 line loop - carry estimated error along
  - can sum 100K values, get ~1% error in sum, sufficiently often
    (often with higher levels of software further compensating)
  - usefulness shown in following examples

RICHARDSON LUCY DEBLUR

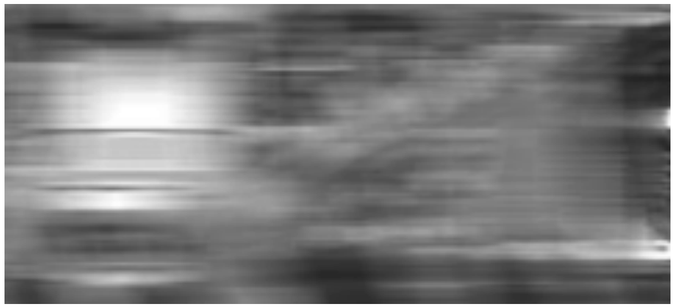
original

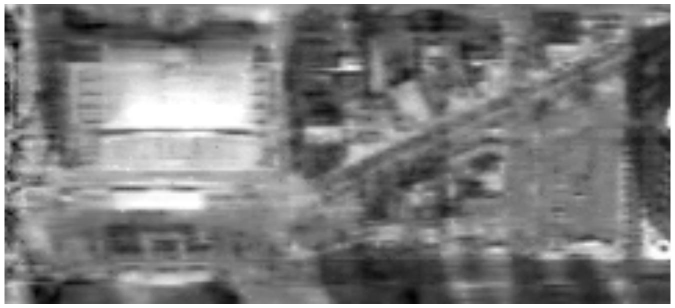




ieee fp

blurred





1% fp

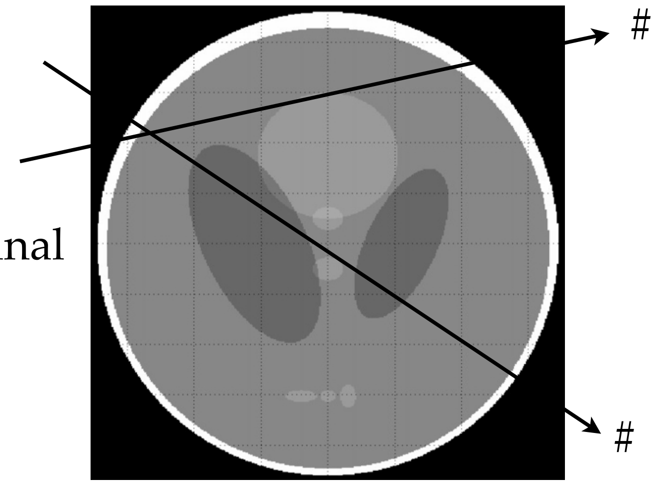
Stack = Kahan + iterative descent on error

(believe higher level can solve for error alone in similar manner,
accuracy becomes 1% of 1% ~ perfect)

TOMOGRAPHY

further evidence that lots of arith can work

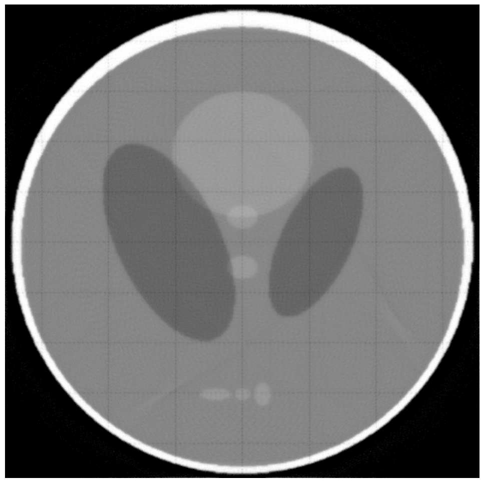
original

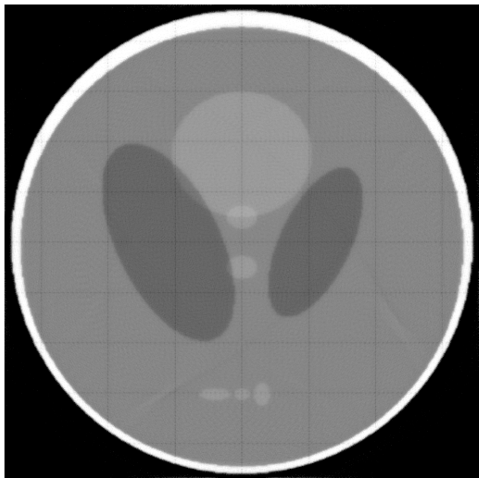


"iterative reconstruction"

results here used
~ 100G arith ops

ieee fp





1% fp

NEAREST NEIGHBOR

- Use brute force method (database on chip, search all in parallel)
  - example: short 5-vectors, from N(0,1) distribution
    - if chip finds best <u>one</u> - 95.6% correct
    - find best <u>two</u> (then CPU chooses) - 99.7% correct
  - stack = Kahan + find several candidates, let CPU pick

  ⇒ *more evidence sw can derive high quality results from approximate hw*

- Notes:
  - brute force => works in high dimension
  - some algorithmic cleverness usable, eg hashing
  - can efficiently stream large database through in chunks, if have enough simultaneous queries

    (chunk loading cost amortizes over query cost)

OTHER PROMISING DOMAINS

- Physical sim - if system robust to underlying physical noise
  - molec dynamics, protein folding (thermal noise, models approx)
  - electrical sim of digital circuits (silicon noise, fab errors)
- Machine learning, when data noisy, learned models approximate
  (eg neural net training works, now exploring graphical models)
- Numerical optimization and some combinatorial opt (int/bool support)
  - run 100K starting points in parallel
  - clean up best ones w/accurate fp math (on chip or on cpu)
- Image processing at low power/size
  (small autonomous vehicles, cameras, mobile video)

PATH FORWARD

- Hardware ready
  - everything innovative is designed, simulated, verified
  - surrounding hardware familiar, easy
  - working with chip design firm to be sure
  - chief architect of 4 Intel Pentiums - thumbs up in DARPA review
  - IP protection in place, to aid commercial scale-up, to enable science

- Software must be explored far more widely
  - it's where risks and opportunities lie

- Chicken and egg:
  - to get the benefits and advance their field, scientists need hardware
  - but cheap hardware follows proof of wide application by scientists

- Currently collaborating with Deb Roy at Media Lab
  - large scale video analytics (tracking) with ONR funding
  - exploring software - testing code using hardware emulator

- Next goal - get real hardware out to multiple scientific groups

  But silicon fabrication costs very nonlinear
  - $1M not helpful if want large machines
  - $4M yields 10 machines, each with a **<u>million</u>** cores (PEs)

  So goal: spend ~$4M, seed ~5 universities/government contractors
  - explore varied domains, e.g. vision, image processing, learning, speech, biology/medicine, other computational science
    + offer free and open access for students and other faculty
  - share basic tool development, code libraries, experience

- If results promising, seek large company(s) to scale up production, bring down prices, make available to broad research community