Zooming out and scaling up: from 1 to 1000 cores

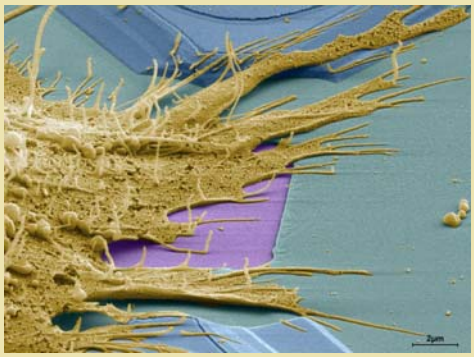
Steven P. Levitan
University of Pittsburgh
Thanks to Donald Chiarulli

How we got (back) here

Two technology drivers to renewed interest in parallel processing:

- Heat problems (hot chips)
  - Power: cooling vs. melting
    - Slower clocks
  - More instructions per clock
    - More instruction interpreters
- Nano (bio?) technology
  - Billions and billions of devices
  - None of them working very well

http://www.devilsfoot.com/Power/Pics/blv910die.jpg



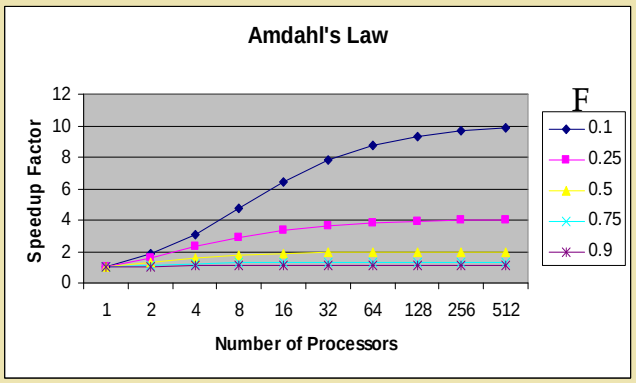
http://www.fz-juelich.de/ibn/datapool/page/397/CellOnTransistor.jpg

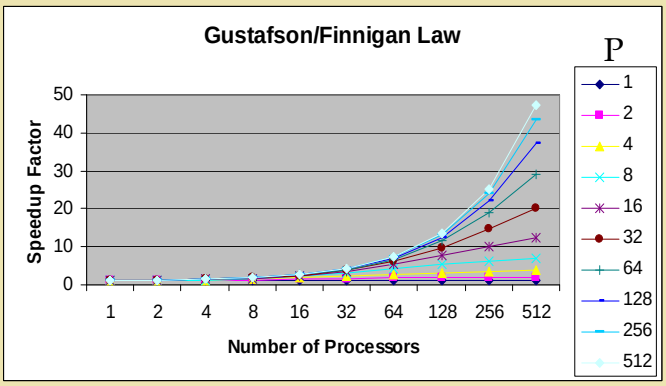
How fast can we go?

- Amdahl 1967: speedup factor for a fixed size problem
- F = Fraction of "non-parallel code"
- N = number of processors

Gustafson 1988: scale the problem size

- C = Constant amount of serial code
- P = problem size
- N = number of processors





Speedup = 1 / (F + (1-F)/N)

Speedup = (C+P) / (C+(P/N))

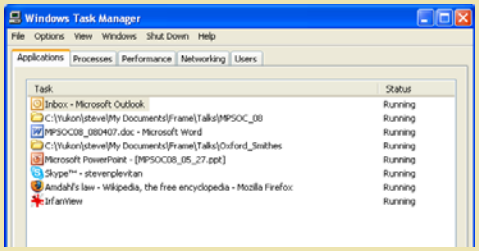
The VLSI approach to computational complexity[1] – J. Finnegan

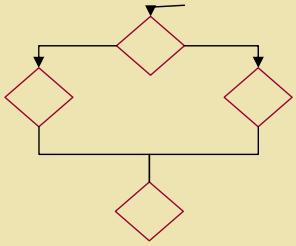
- "The rapid advance of VLSI and the trend towards the decrease of the geometrical feature size, through the submicron and the subnano and… beyond … [implies that] traditionally unsolvable problems can now be easily implemented using VLSI technology."
- "…in the 70s … scientists discovered… [that] problems with [high] time complexity can be solved in [little] time… using a number of processors which is a function of the problem size..."
- "…the cost of VLSI processors decreases exponentially. … Hence, the application of an exponential number of processors does not cause any cost increase…"

[1]D. Cohen in "CMU Conference on VLSI Systems and Computations," Kung, Sproul, Steele ed. Computer Science Press, Rockville, MD., pp. 124-125 1981

What works?

- Few Cores
  - Task based
- A Bunch
  - Multi-threading
- Many (Streams)
  - Data Decomposition
  - Problem Partitioning
- A lot
  - New Paradigms

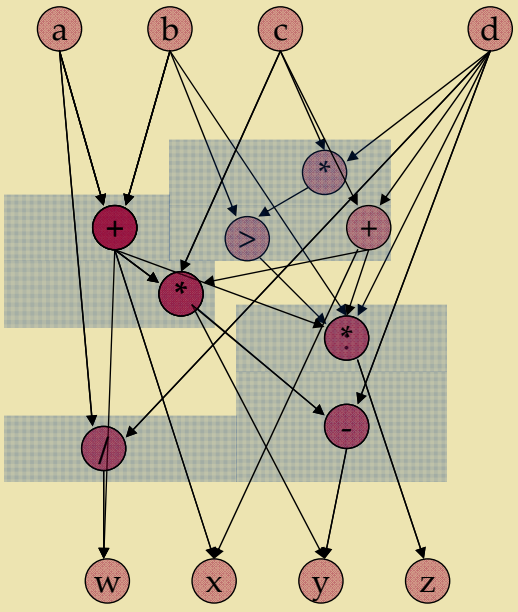




Beyond Flynn's Streams: a continuum of architecture models

Streams of instructions, operations, data, sequencing

- Dataflow
  - Directly compute the graph (forest)
  - No program counter
- VLIW
  - Several different instructions at once
- MIMD
  - autonomous behaviors
- SIMD
  - same program in lockstep
  - data dependent branching done by bifurcating execution
- Cellular Automata – same program – but not broadcast



Classic parallel paradigms

Q: How to write parallel programs?

A: Decompose the problem by:

- Space
  - Map data representation of problem to processors
  - 3D or could be problem space, solution space
- Time
  - Pipelining partial results
- Function
  - Task assignments to processors
- Instance
  - Multiple instances of problem on processors

What did we learn from VLSI era parallel processing?

- Hardware is the easy part
  - Broadcasting, streaming, synchronizing, counting, etc.
  - Memory – you need more than you think
  - Serial process (remember Amdahl's Law)
  - I/O bottlenecks can kill performance
  - Scaling problem for global communication
    - End to end delays grow as the mesh size scales up, O(d)
- Software is the hard part
  - We are taught to think in a linear sequence
    - Not just for writing programs
  - Discovering parallelism after the creation of the algorithm is hard for both humans and machines

Higher level *metaphors* for computation

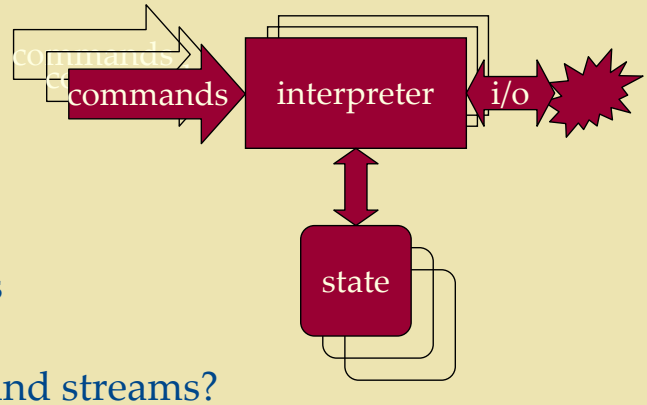
- Interpretation – reacting to commands
  - Layers-abstractions
  - Control systems, operating systems
- Transformation – translating representations
  - Data analysis
  - Communication
- Simulation – modeling systems
  - Physical systems
  - Artificial systems
  - Social Systems
- Optimization – exploring state-spaces
  - Representations
  - Processes
- Are these are the right ones, are there more?…

3D Life

http://www.geocities.com/robisais/3dca.html

Interpretation

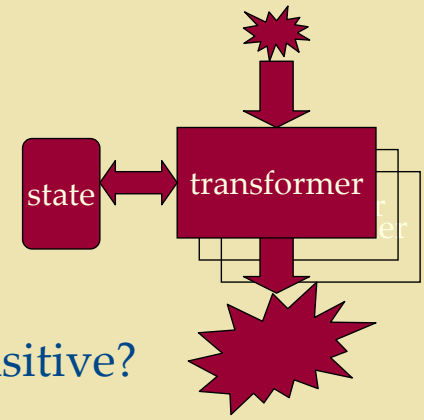
- Examples:
  - operating system
  - robotic control
  - human interaction
  - stock market
  - airline reservations



- Design Issues:
  - How many command streams?
  - Do we have limited resources
    - scheduling
  - Shared or independent state spaces?
    - Synchronization
  - Response time

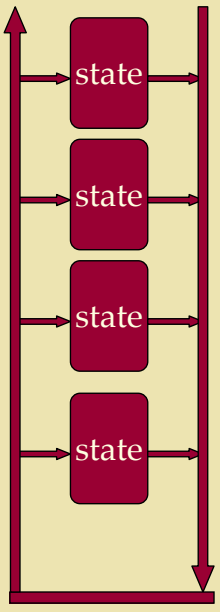
Transformation

- Examples
  - Compilers
  - Image/signal processing
- Design Issues
  - Is the input context free/sensitive?
    - Manage global state
  - Is the change incremental?
  - Does the size of the representation change?
  - Throughput



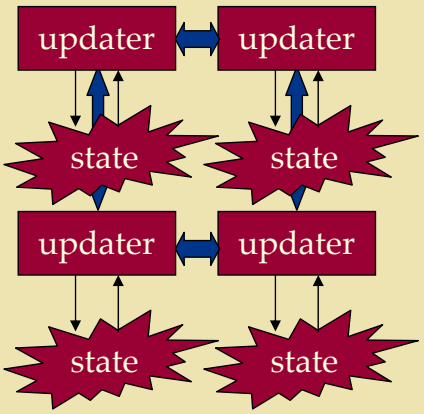
Simulation

- Examples
  - Discrete time systems
  - Economic, social systems
- Design Issues
  - Time management
    - scheduling
  - Number of actors
  - Interaction network
  - Evaluation complexity



Optimization

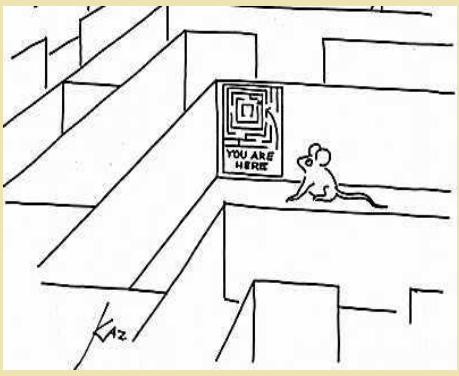
- Examples
  - Branch & bound
  - Genetic algorithms
  - Relaxation
- Design Issues
  - Distributed vs. global state
  - Cost/optimization function



Work in the problem domain

Rather than try to "parallelize code"

- Concurrency should be intrinsic to the representation of the problem
- Use a language that correspond directly to the concepts in the problem domain
- The ideal language should present both
  - "rats eye view" - neighborhood
  - "birds eye view" – overall behavior



www.cartoonstock.com

Think Globally, Act Locally