UNIVERSITY of VIRGINIA

Implications of the Power Wall in a Manycore Era

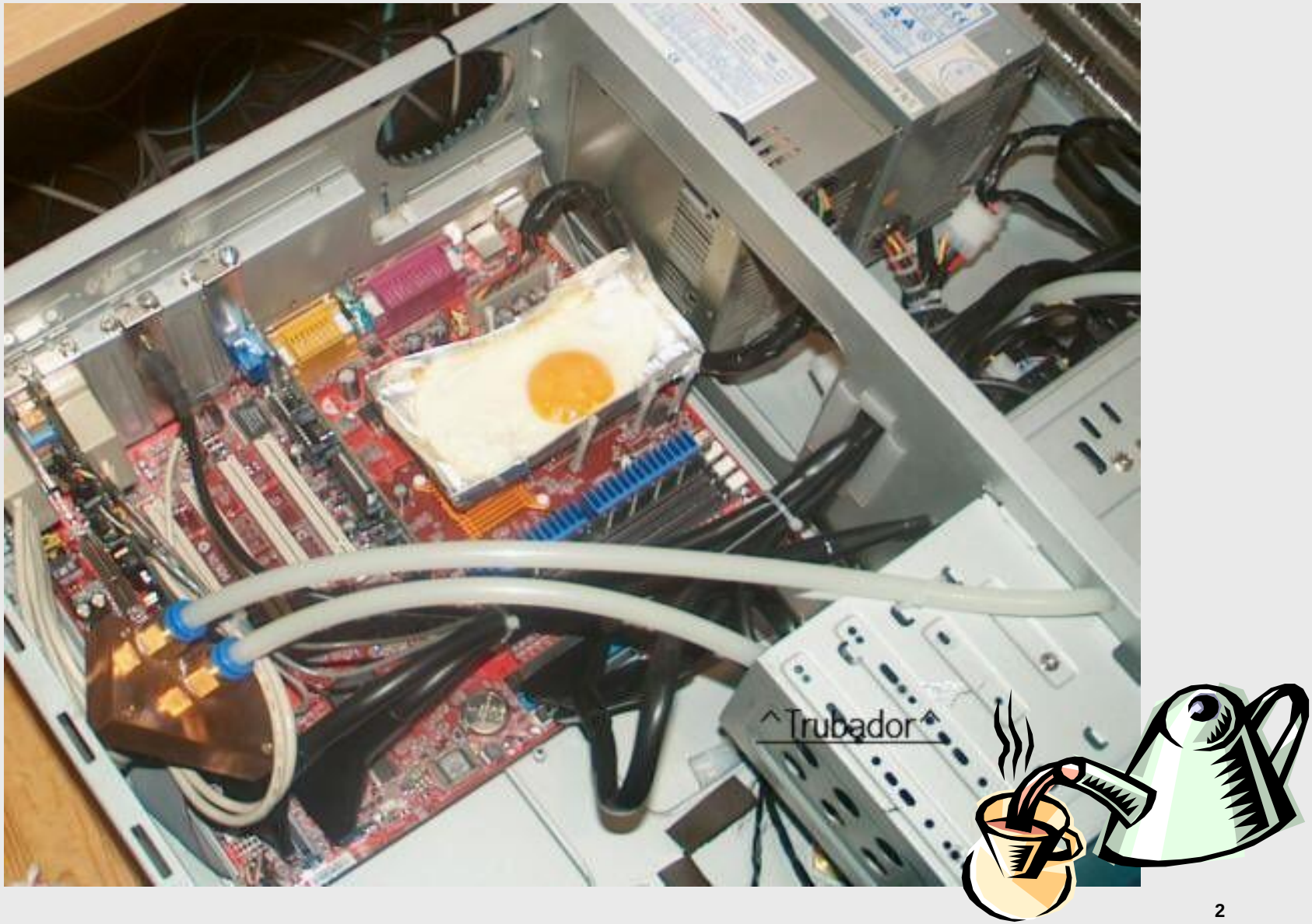
Sustaining Growth in Computing Performance

Kevin Skadron

LAVA/HotSpot Lab, Dept. of Computer Science
University of Virginia

Visiting Professor
NVIDIA Research

“Cooking-Aware” Computing?



Outline 1

- Summary of barriers to performance growth
  - Barriers, implications
- Driving applications
  - Which aspects of system performance will most be in demand?
- Main body of talk
  - How can these needs be met?
  - What are the models of computing in the future likely to look like?
  - Role of power

Barriers to Performance Growth (1)

- Memory wall
  - How to ramp up memory bandwidth as we ramp up # cores
- Variability
  - Within-die, *core-to-core* variations due to PVT
    - DVS, RBB can't completely eliminate
    - Speed determined by worst core
    - Or software must countenance variety of core speeds
- Reliability challenges
  - Defects and errors of all types (including due to PVT variations) more likely at smaller feature sizes
  - Making good progress on this front, must keep up research
    - Modular designs for flexible redundancy, soft error detection and correction, etc.
    - Decades of research on this topic to draw on
    - Research needs to track most likely future manycore architectures
    - Need to get down power, performance cost

Barriers to Performance Growth (2)

- Power wall, ILP wall, and slow frequency growth combine to force multicore
  - Limited single-thread performance: slow growth but not doubling every 18-24 mos.
  - *No longer cost effective to dedicate increasing transistor count to ILP extraction for single-thread performance*
  - More aggressive ILP extraction techniques (kilo-instruction, TRIPS, etc.) are promising but too late
    - Too long, perhaps too risky to put into production
    - Industry is already committed to multicore
    - On-chip integration is a sweet spot
      - Low intra-core communication is a paradigm shift for parallel programming
  - *Power wall will eventually constrain multicore too*
    - If power per core doesn't scale down as number cores scales up
- Programmability: parallel programming is hard
  - Major productivity concern

Why Do We Care?

- Important, long-term social concerns exhibit staggering opportunities, computational demands
- Rapid performance growth is now a fundamental growth driver
  - for economic growth
  - for standard of living
- Examples
  - Medical discovery, diagnosis
    - Example: cancer drug screening – maxed out multiple clusters across UVA for months
    - Example: leukocyte detection and tracking in video microscopy for inflammation studies: minutes/frame
      $\Rightarrow$ no real-time feedback possible
  - Scientific discovery—massive data processing, modeling, simulation
    - Energy issues and basic science supporting medicine (e.g. protein folding) are of particular economic significance
  - Commercial: E-commerce, financial analysis, communications/telepresence, multimedia, etc.
    - Vitally important to the economy
  - Security, privacy, sensor nets, etc.
- Lots of parallelizable, scalable, big-$$$ and national-interest applications
  - These will get us over the initial hump of developing a market and ethos for parallel programming

Where We are Today

Nasty side effects of scaling

Voltage-based techniques

Excess hardware in the core, e.g. ILP discovery, very deep pipelining

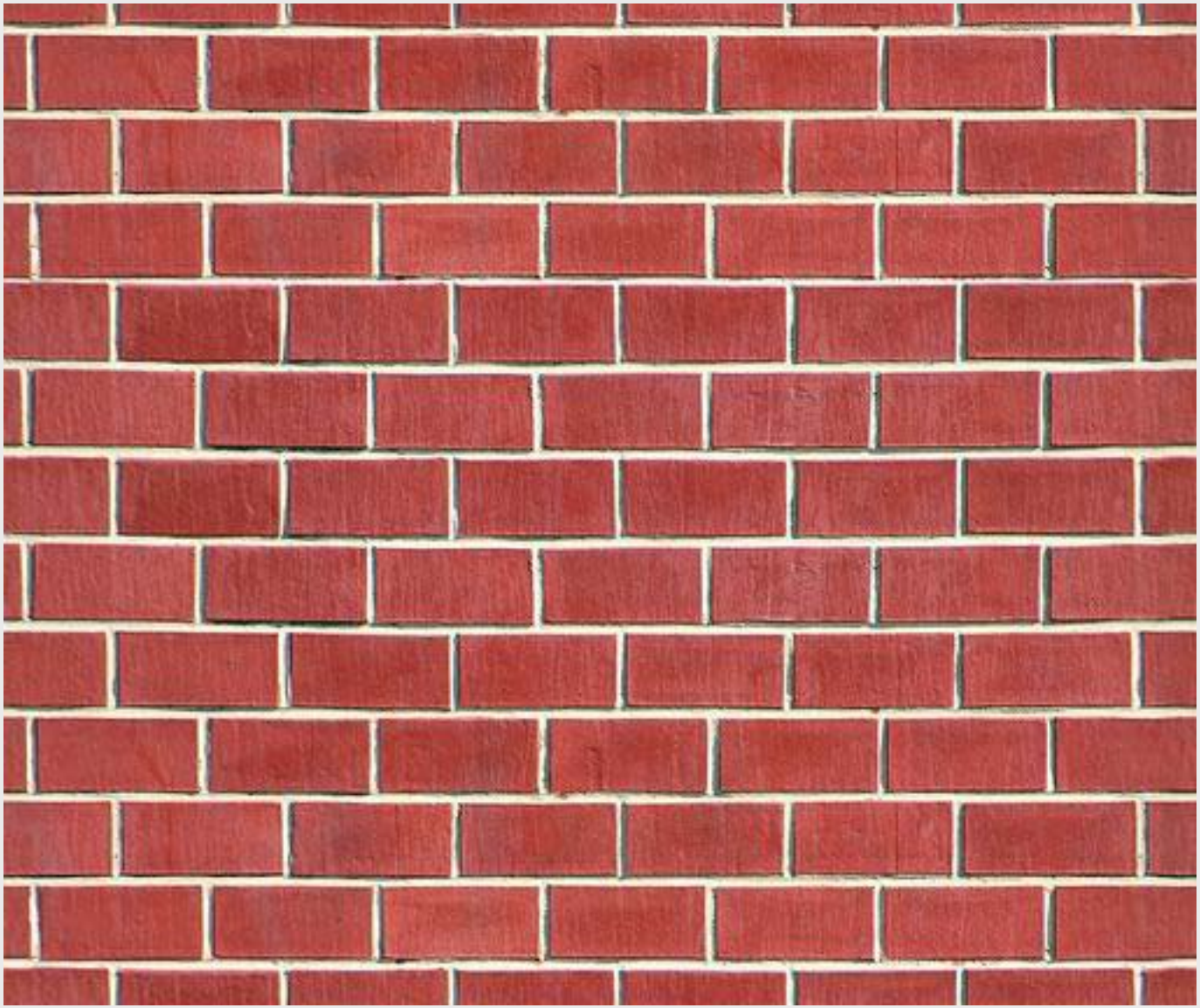
Big Picture

- **Multicore is a one-off (fortunately, a big one)**
  - **The power wall is still there**
  - **Can't keep packing more and more conventional cores at current clock frequencies**
- **At the same time, normal mortals can't program massively parallel architectures**
  - **How can application needs best be met?**
  - **Must adopt new abstractions, supporting architectures, new algorithms**
- **Shift to multicore + new abstractions create a rare opportunity to transform hardware *and* software**
  - **Release hardware from the ISA death grip and let it evolve with technology**

Outline 2 (Rest of the Talk)

- Technology trends (why multicore isn't sufficient to skirt the power wall)
- Implications of the power wall for core architecture + research challenges
  - *TDP-scalable* macro-architectures
- Programmability challenges and implications
- Where do we go from here?
  - Need to invest heavily in higher-level abstractions for parallel programming
  - Ensure that new programming models are compatible with expected multiple scenarios for HW evolution
  - Explore scalable architectures
  - Keep attacking the memory wall, etc.

The Power Wall

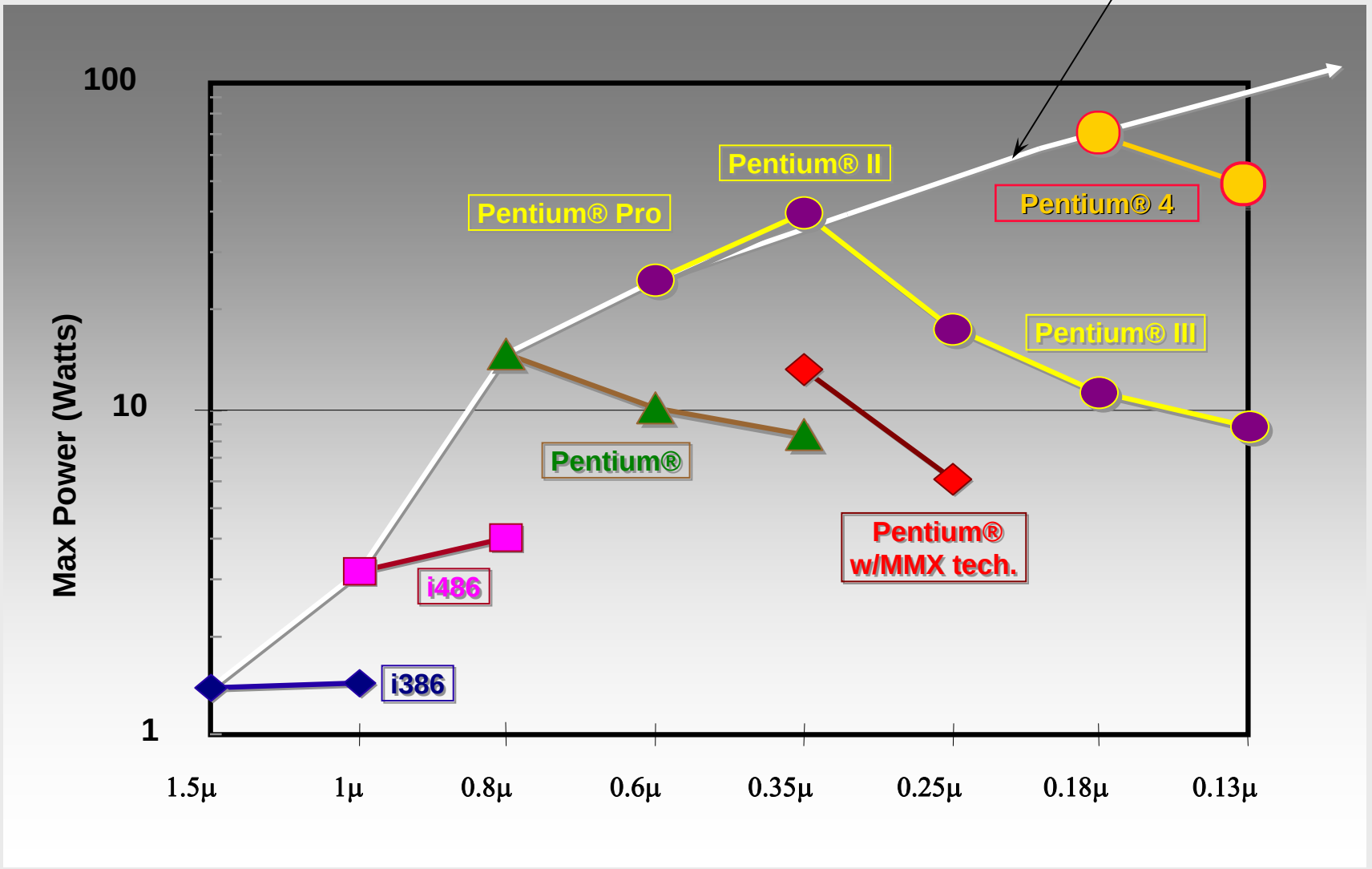


http://mraybould.wordpress.com/2007/05/28/if-you-are-mortar/

Moore's Law and Dennard Scaling

- Moore's Law: transistor density doubles every N years (currently N ~ 2)
- Dennard Scaling
  - Shrink feature size by k (typ. 0.7), hold electric field constant
  - Area scales by k^2 (1/2) , C, V, delay reduce by k
  - $P \cong CV^2f \Rightarrow$ P goes down by k^2
- We never saw this in practice
  - Generally kept area constant, used doubled transistor density to add more features, so C didn't scale
  - Aggressive pipelining, circuits, etc. to boost f beyond "natural" rate
  - Leakage
  - Power and power density went up, not down
- "Natural" frequency growth may slow even further due to increasing parasitics, parameter variations, etc.
  - Projections I've seen suggest frequency growth of 15-20%/year, not clear if even this is sustainable

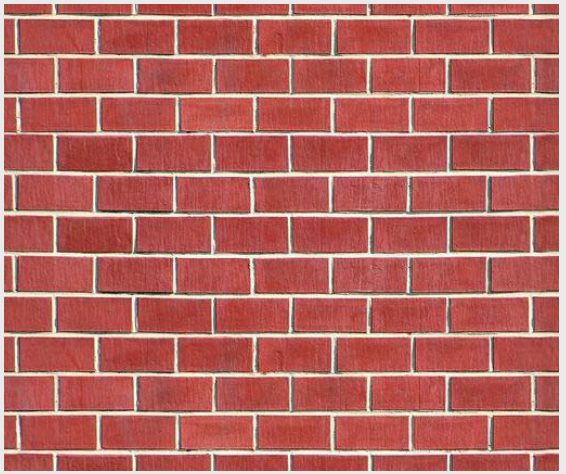
Actual Power



Source: Intel

Why Multicore

- **Why multicore?**
  - **Power wall + ILP wall = brick wall**
  - **Continued scaling of feature *sizes* seems secure for many generations**
  - **Parallel programming more palatable on same chip**
- **But the power wall is still there**



Vdd Wall

- Vdd will scale slowly, if at all
  - 2008-2011: 1.0V (ITRS 2006, high-perf)
  - 2012-2014: 0.9V
  - ...2020: 0.7V
  - This is about 2.5%/yr
- To maintain suff. drive current, Vdd must be suff. larger than Vt
  - Lowering Vt, combined with short-channel effects, makes leakage grow exponentially
- Vdd floor isn't scaling either
  - ITRS allows only a 0.2-0.3V margin
- Doubtful that voltage domains solve this problem
  - Fine-grained domains difficult to route in high-power chips
  - Doesn't help power density in high-perf/high-power domains anyway

Power Wall Redux (fundamental limits)

- Even if we generously assume C scales and frequency is flat
  - $P \cong CV^2f \Rightarrow 0.7\ (0.975^2)\ (1) = 0.66$
- Power *density* goes up
  - P/A = 0.66/0.5 = 1.33
  - And this is very optimistic, because C probably doesn't scale so well, so a more likely number is 1.5-1.75X
- If we keep area dedicated to all the cores the same, total power goes up by the same factor
- But cooling capabilities aren't scaling so fast
  - ITRS holds max power fixed at 198 W for high-performance, 310mm^2 die for 2007-2020
    - This is a cooling, not a power-delivery constraint
  - 104-137W at 140mm^2 for cost/performance (5% growth)
  - Power density fixed at 0.64 W/mm^2 (high-performance), up to 0.98 W/mm^2 for cost-performance
  - We are near affordable air cooling limits
- Battery capacity isn't scaling fast enough either
- Also limited by max amperage of a household wall outlet!

Implications

- Single-core frequency scaling will be slow
  - But beware Amdahl's Law
- Multi-core scalability faces the power wall soon
  - If multicore scaling slows down, the only way to improve performance is with multiple chips/computers
    - Multicore-driven adoption of parallel programming will help in the multicomputer realm
    - But long communication delays, memory-system issues make these harder to program, harder to scale
  - Scalable, single-chip performance is the sweet spot
    - Need to squeeze as much out of the chip as possible

What Power Efficiencies Need to Scale?

- Need to minimize power per bit of I/O and on-chip communication
  - This is important because the core no longer dominates power as much
  - And because there are so darn many cores
- Maximize reuse
  - Avoid repeated roundtrips to L2 or main memory
  - Try to keep producer-consumer chains on same core or between nearby cores
  - Requires dynamic load balancing
    - Multithreading helps
  - Software ordering for locality helps
- Need *TDP-scalable cores*

Low-Fat Cores

PClaes Oldenburg, *Apple Core – Autumn*
http://www.greenwicharts.org/pastshows.asp

Low-Fat Architecture (What power efficiencies need to scale?)

= research priority

Same single-threaded performance

- Make ILP cores more power-efficient, more low-power modes
  - Running out of steam
- Heterogeneous organization (same area): 1-2 ILP cores + scale up #simple cores
  - Combines problems of high-power cores and multiplying number of cores (TDP + hotspots) – esp. if we want the ILP core to be really fast, it is the thermal limiter
  - Dynamic cores (composing powerful single-threaded cores out of "throughput cores") is a better way to support single-thread perf => retain homogeneous hardware
- Heterogeneous organization (MCM)
  - Let chip size scale down; achieve throughput scaling with multiple (possibly heterogeneous) chips – tightly coupled
  - Power costs for core-to-core communication, B/W limitations?
- Specialized coprocessors or functional units
  - Special-purpose cores are way more energy efficient + spread activity out
  - Only help for apps that can use them; dubious area efficiency
  - Programmable coprocessors offer wider flexibility albeit lower efficiency

Throughput focus

- Amortize hardware in time (ie, multithreading)
  - Exploit memory parallelism -- great throughput/W benefit up to a point (cache size becomes huge with multithreading)
  - Can be 1) transparent using demand fetch (e.g. Niagara, GPUs), or 2) [illegible] software-managed computation stages overlapped w/ DMA (e.g. Cell)
  - But (1), thread count keeps growing, and (2), dynamic memory-access patterns are challenging
- SIMD – amortize fetch/decode/etc. hardware in space
  - Divergent code is a problem
  - Only effective for limited application space???
    - Maybe not, with proper abstraction layers and MIMD set of SIMD cores
  - Long-term scalability??? Simpler ALUs?

Thermal characteristics

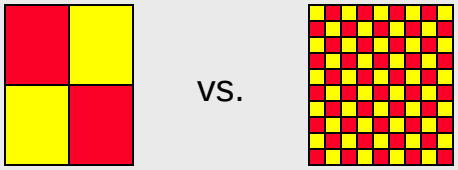
Programmability

Refining the previous argument

- **Mark Horowitz summarized my argument as follows**
  - **Energy per op is a non-linear function**
  - **We were spending lots of E for modest extra performance (ILP, extra frequency)**
  - **Right now we are scaling down E/op with modest performance cost**
    - **This lets us keep packing on more cores within an acceptable power budget**
  - **Once we hit the optimum E/op, further reductions in E/op will have a high cost in performance**
  - **Not clear what to do next**
    - ***Are we stuck???***

Thermal Considerations

- **Cooling is the main constraint**
  - **Pick max Tj, typically 100-125C, based on reliability, leakage tolerance, and ergonomics**
  - **The most thermally efficient design maximizes TDP (and hopefully throughput) under this constraint**
    - **Hotspots hit Tj faster => thermal non-uniformity represents a lost opportunity**
- **Seek thermally uniform macro-architectures**
- **Multicore layout and "spatial filtering" give you an extra lever**
  - **The smaller a power dissipator, the more effectively it spreads its heat**
  - **Ex: 2x2 grid vs. 21x21 grid: 188W TDP vs. 220 W (17%)**
    (very preliminary result)
    - **Increase core density**
    - **Or raise Vdd, Vth, etc.**



  - **Thinner dies, better packaging boost this effect**
- **Seek architectures that minimize area of high power density, maximize area in between, *and can be easily partitioned***

TDP-Scalable Architectures

- Can we make conventional cores small enough?
- Which is better: more, specialized functional units, more threads, VLIW, SIMD…?
- Example: SIMD
  - \+ Amortizes fetch, decode, control, and register-access logic
  - \+ Tends to better preserve memory-access locality
  - \+ Space savings allow more ALUs or on-chip memory in same area/total power
  - \- Tends to have nasty crossbars
  - \- Doesn't deal well with threads that can't stay in lockstep
    - Multiple cores of limited SIMD width
    - Work queues, conditional streams, etc. needed for reconvergence
  - \- How to support single-thread performance?
    - Processor for a single "thread" is typically pretty wimpy
  - \- Densely packed ALUs
    - Can they be spread out?

Outline

- Technology trends (why multicore isn't sufficient to skirt the power wall)
- Implications of the power wall for core architecture
- **Programmability challenges and implications**
- **Where do we go from here?**
  - **Need to invest heavily in higher-level abstractions for parallel programming**
  - **Ensure that new programming models are compatible with expected multiple scenarios for HW evolution**
  - **Explore scalable architectures**
  - **Keep attacking the memory wall, etc.**

If We Build It (multicore), Will They Come?

- Claim: programmers who can do low-level parallel programming are an elite minority
  - We will never train the "average programmer" to write highly parallel programs in C, Java, X10, Chapel, CUDA, etc.
  - People need to think about things sequentially
    - Requiring programmers to reason about too many independent "things" won't work
      - Programmers are also put off by extensive setup, bookkeeping
    - But it's ok if the "things" are internally parallel
    - Also a good model for dealing with heterogeneous chips, networked ensembles, etc.
- Must develop APIs with higher-level abstractions
  - We need this regardless of what the underlying architecture is
  - But it also buys us more flexibility in the architecture
  - DirectX is a good case study
  - Need more domain-specific languages/APIs

DirectX

- High-level abstractions
- Serial ordering among primitives
  = implicit synchronization
- No guarantees about ordering within primitives
  = no fine-grained synchronization
- Domain-specific API is convenient for programmers and provides lots of semantic information to middleware: parallelism, load balancing, etc.
- Domain-specific API is convenient for hardware designers: API has evolved while underlying architectures have been radically different from generation to generation and company to company
- Similar arguments apply to Matlab, SQL, Map-Reduce, etc.

➢ I'm not advocating any particular API, but these examples show that high-level, *domain-specific APIs are commercially viable and effective in exposing parallelism*

- Middleware (hopefully common) translates domain-specific APIs to general-purpose architecture that supports many different app. domains

Where We are Today - Multicore

Programmability wall

Power wall

The one language that will rule them all

Von Neumann model?

http://interactive.usc.edu/classes/ctin542-designprod/archives/r2d2-01.jpg

HW Implications of Abstractions

- If we are developing high-level abstractions and supporting middleware, low-level macro-architecture is less important
  - Look at the dramatic changes in GPU architecture under DirectX
  - If middleware understands triangles/matrices/graphs/etc., it can translate them to SIMD or anything else
  - HW design should focus on:
    - Reliability
    - Scalability (power, bandwidth, etc.)
    - Efficient support for important parallel primitives
      - Scan, sort, shuffle, etc.
      - Memory model
      - SIMD: divergent code – work queues, conditional streams, etc.
      - Efficient producer-consumer communication
      - (These primitives might turn into new coprocessors or functional units)
- Analogy to the RISC revolution
  - The primitive ops are the new ISA, and current ISAs become the new microcode/micro-ops
  - "ISA" supports the compiler/middleware, not the programmer
- Need to support legacy code (MPI, OpenMP, pthreads)
  - Low-level APIs should work with these codes
  - This will also allow programmers to "drill down" if they need to

Top Research Challenges

- Develop domain-specific APIs and portable, common middleware
  - Make sure these primitives and APIs are compatible with multiple hardware evolution scenarios (heterogeneous cores, SIMD, etc.)
  - Use existing low-level APIs for now (CUDA, TBB, CTM)
  - Need ecosystem of portable profilers, debuggers, etc.
- Explore TDP-scalable manycore architectures, focus on evolutionary strategies
  - Can develop APIs and use them today!
    - GPUs, Fusion, Larrabee, Cell, Niagara, etc.
    - Radical architectures can teach us about limits, but unlikely to gain market adoption due to engineering effort
  - Need new research on how to simulate and evaluate these architectures
  - Need to understand which components in the core scale nicely in power, which do not
- On-chip and off-chip communication bandwidth, latency, and power remain key challenges
- Work with real *programmers* (not just real programs)
  - Maximizes likelihood that outcome is practical
  - Helps speed adoption
  - Helps advance scientific discovery along the way

Summary: Start Investing Now

- **Both APIs and TDP-scalable architectures**
- **Can do a lot with existing platforms and APIs**
  - **GPUs are especially attractive today as massively parallel chips with mass market economics and useful low-level APIs**
  - **Nice integration with visualization**
  - ➢ ***Invest now, before programming practice and hardware architectures become locked in!***
- **Need scalable macroarchitecture to put off power wall as long as possible**

Discussion

These are points that came up during Q&A

- **Runtime variations are a serious *programmability* challenge**
  - **True today: thermal throttling**
  - **Worse tomorrow: too hard to predict thermal interactions**
- **Real-time**
  - **Need ways to predict performance on these multicore systems**
  - **Runtime variations, e.g. thermal throttling, make this worse**
    - **I don't personally see an alternative to sacrificing some peak perf**
    - **But now we barely even have acceptable ways to manage these runtime events deterministically in a single core**
- **Dusty deck – presents huge problem**

Backup

- Thanks to David Tarjan (UVA PhD student), who helped me with many aspects of this presentation
- …and to the whole LAVA group, who served as a sounding board for these ideas over the last year