Advancing Computer Systems without Technology Progress

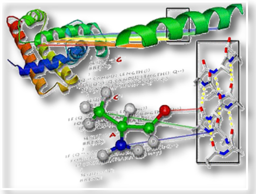
Christos Kozyrakis

Stanford University

http://csl.stanford.edu/~christos

ISPASS Keynote – April 23rd 2013

Computing is the Innovation Catalyst



Science

Government

Commerce

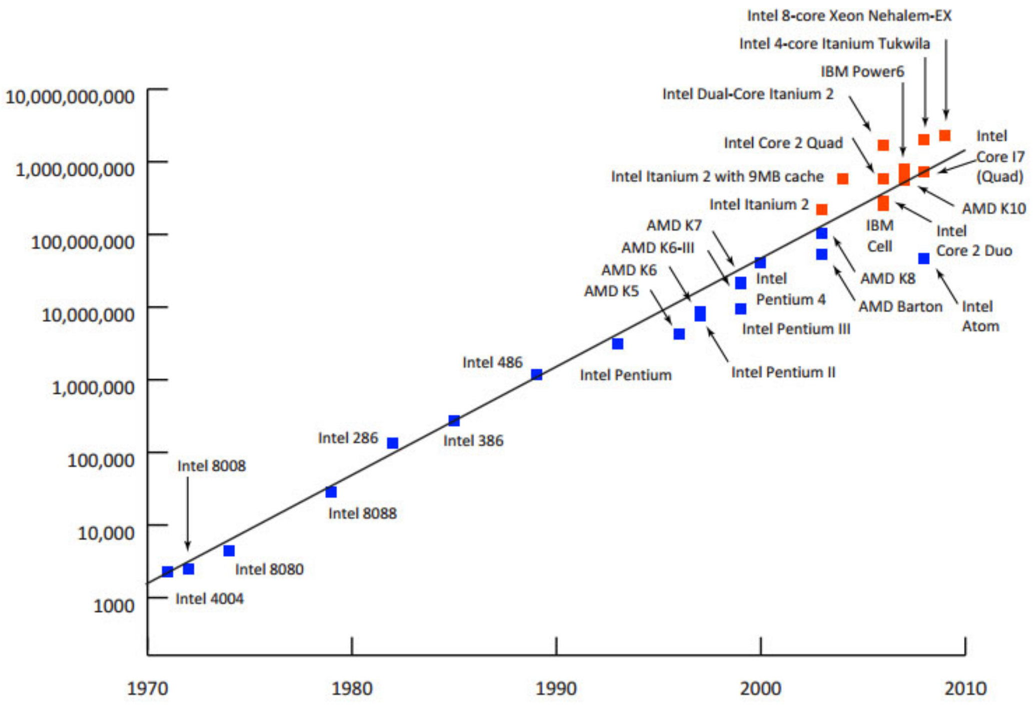
Healthcare

Education

Entertainment

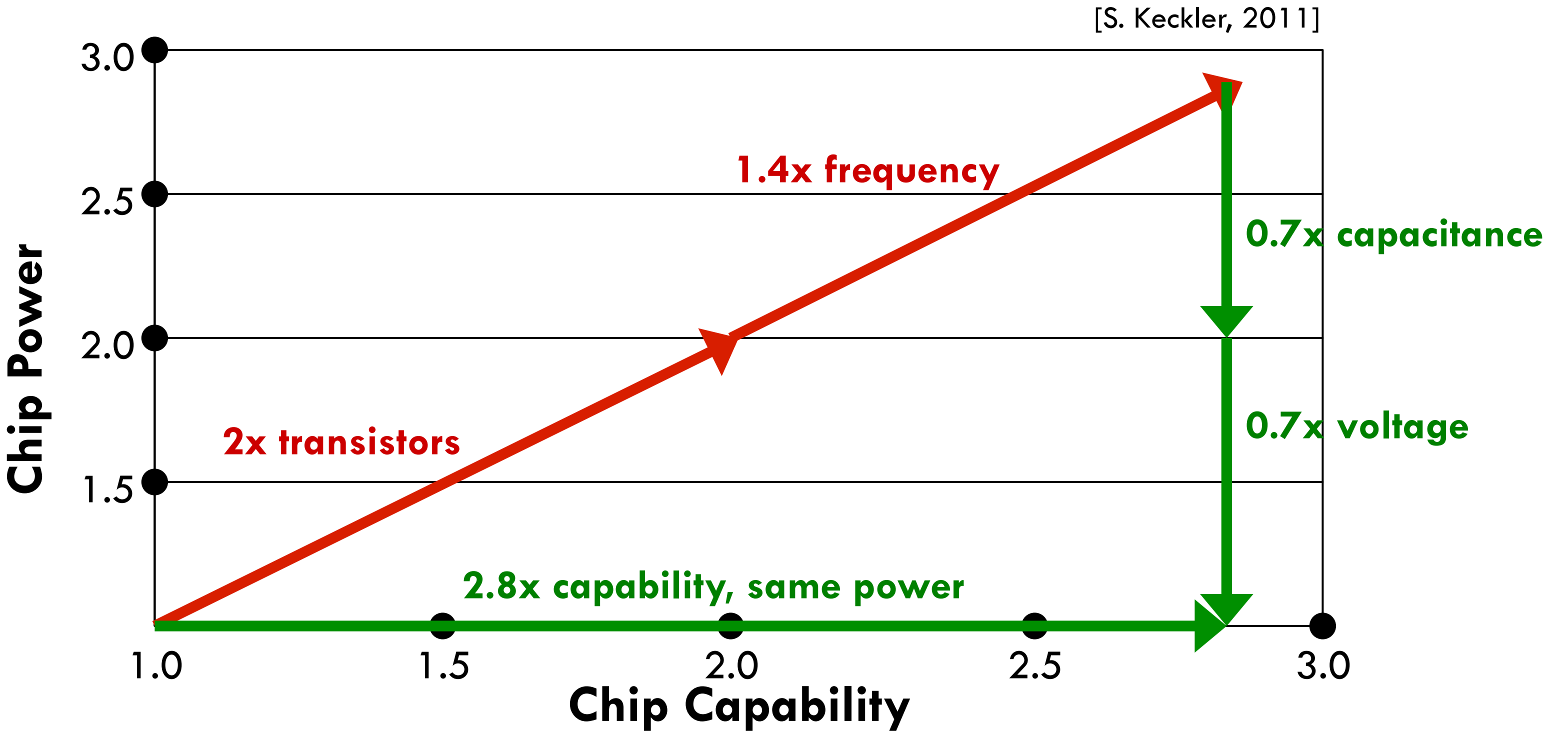
Faster, greener, cheaper

The Key Enabler



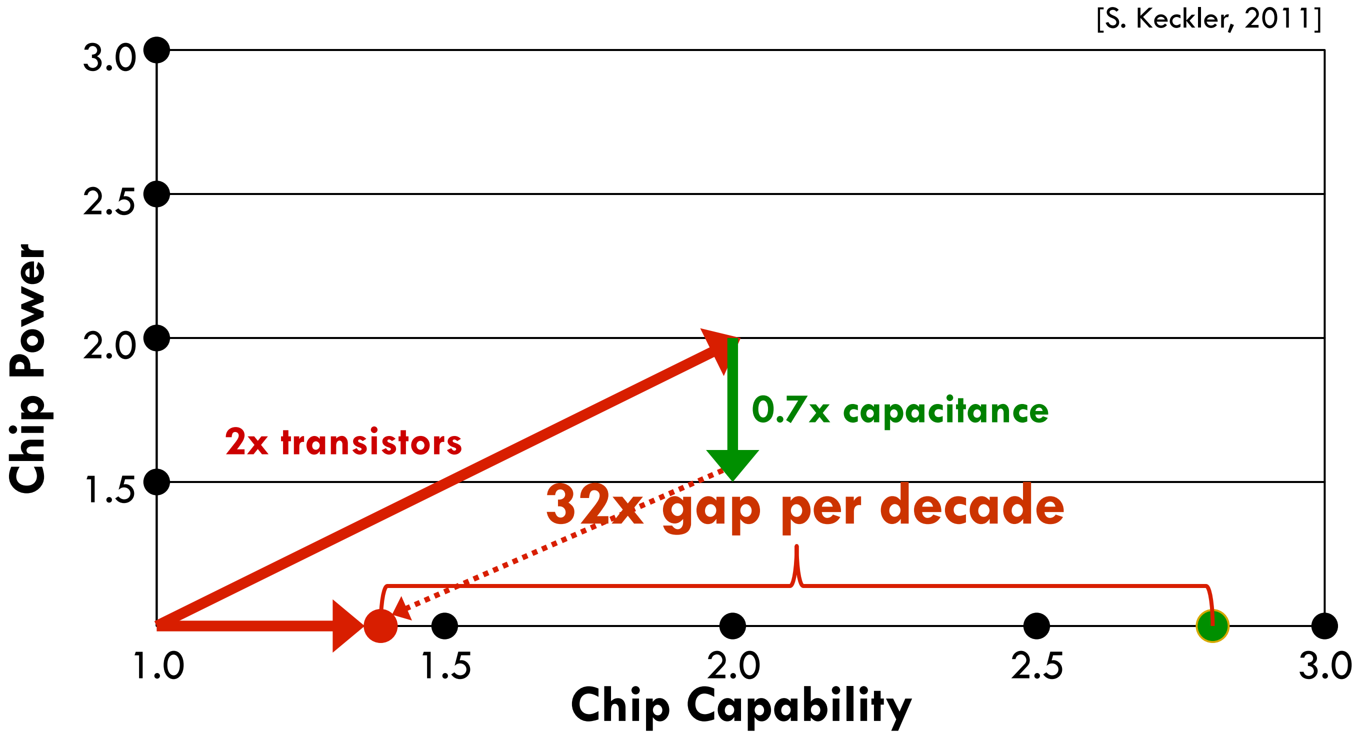
- Turning exponentially increasing transistor counts into
  - Exponentially improving performance
  - At constant cost and power consumption

CMOS Scaling: The Past



- Moore's law (more transistors) + Dennard scaling (lower V_{dd})
  - 2.8x in chip capability per CMOS generation at constant power

CMOS Scaling: The Present

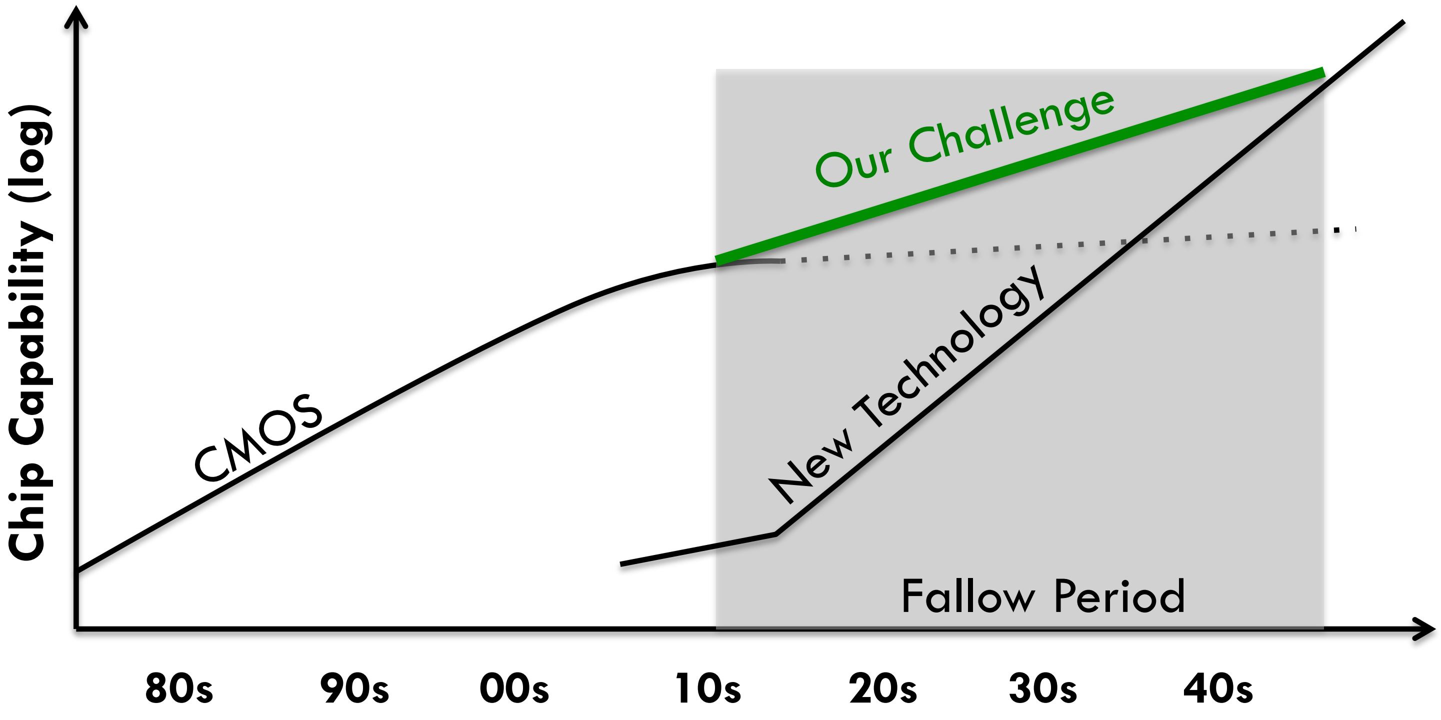


- Moore's Law without Dennard scaling
  - 1.4x in chip capability per generation at constant power
  - 32x capability gap compared to past scaling

Datacenter Scaling

- Cost reduction
  - ~~Switch to commodity servers~~ *one time trick*
  - ~~Improved power delivery & cooling~~ *PUE < 1.15*
- Capability scaling
  - ~~More datacenters~~ *>$300M per DC*
  - ~~More servers per datacenter~~ *@60MW per DC*
  - ~~Multicore servers~~ *End of voltage scaling*
  - Scalable network fabrics

Beyond CMOS?



- Post CMOS technologies are not ready yet
- Need to advance systems without technology progress
  - Need ~3 decades of scaling to cover fallow period

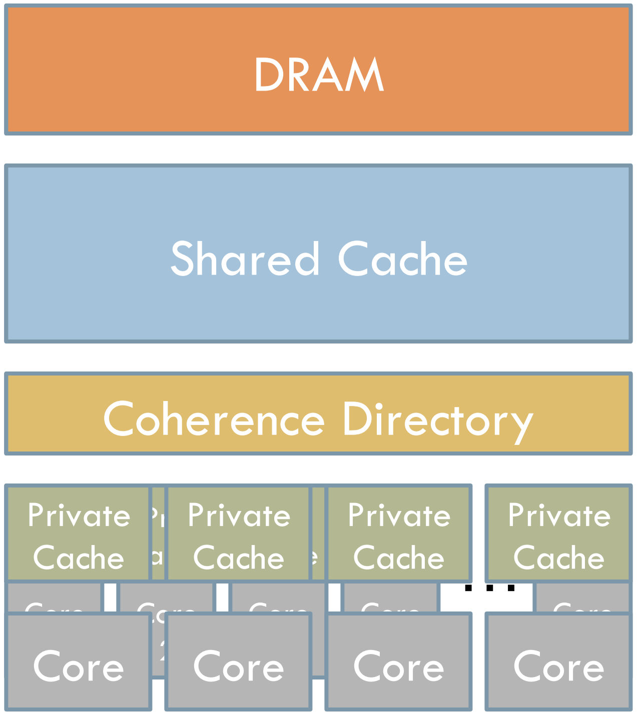
Advancing Systems without Technology Progress

- Locality-aware parallelism
- Specialization
- Reduce overprovisioning
- Increase utilization
- Approximate computing

Advancing Systems without Technology Progress

- Locality-aware parallelism
- Specialization
- Reduce overprovisioning
- Increase utilization
- Approximate computing

Parallelism wo/ Locality → Poor Scaling

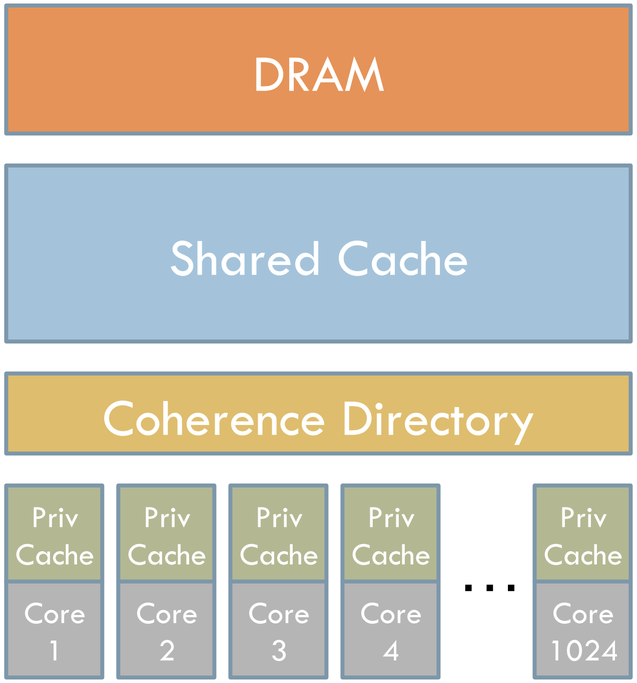


- Memory costs more than compute

16b INT mult:	0.5ns	2pJ
64b FP op:	1ns	50pJ
Shared cache:	10ns	1,000nJ
DRAM:	100ns	10,000nJ

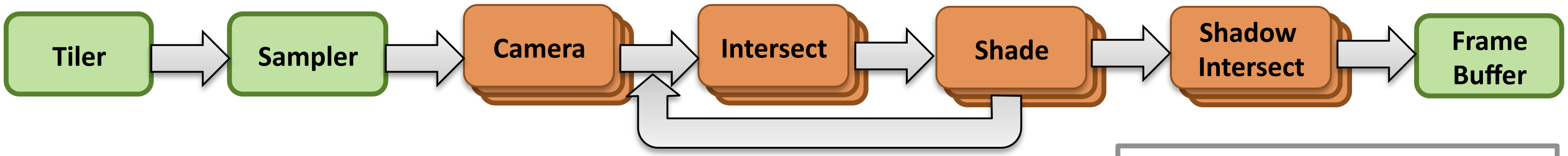
- It will only get worse
  - Poor scaling of long wires
  - Data-intensive applications

Locality-aware Parallelism



- Scheduling for locality
  - Move work to data
- Locality Vs balance Vs overheads
  - Conflicting constraints at large scale

Example: Pipeline Parallelism

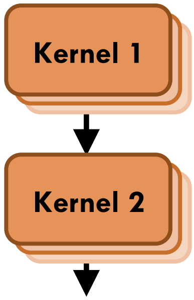


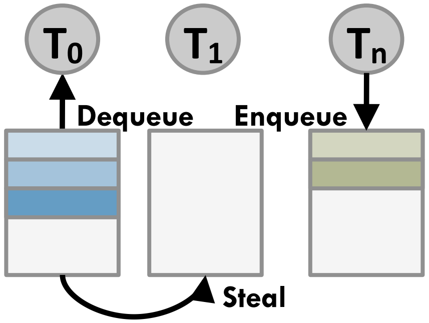
Ray tracing pipeline

- Streaming workloads
  - App as a graph of stages communicating through queues
  - Each stage can be sequential or data-parallel
  - Arbitrary graphs allowed (multiple inputs/outputs, loops)
- Well suited to many parallel apps & compilers

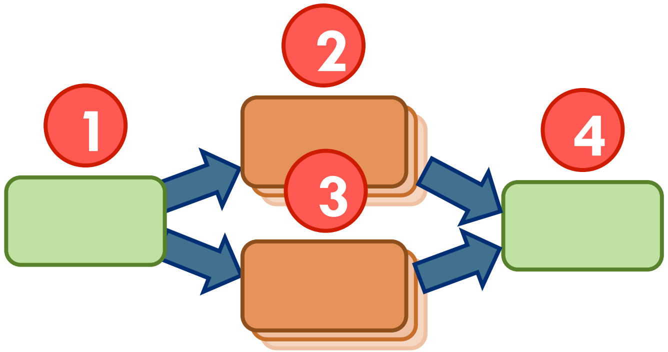
Scheduling Tradeoffs

	GPGPU (CUDA, OpenCL)	Task-stealing (Cilk, X10)	Static (StreamIt, Delite)
Max parallelism	✗	✓	✓
Load balancing	✓	✓	✗
Locality-aware	✗	✗	✓
Bounded footprint	✗	✗	✓
Low overheads	✓	✓	✓



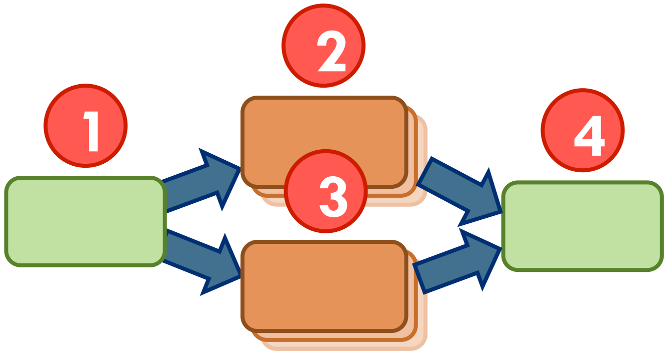


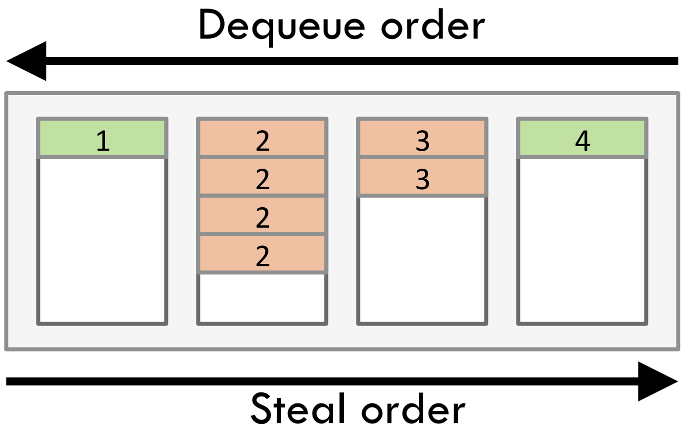
Locality-Aware Dynamic Scheduling



- Insight: use info available in application graph
  - Max parallelism: allow any core to work on any stage
  - Max locality: process intermediate data asap
  - Bounded footprint: account for use queues
  - Load balance: steal task without impacting locality

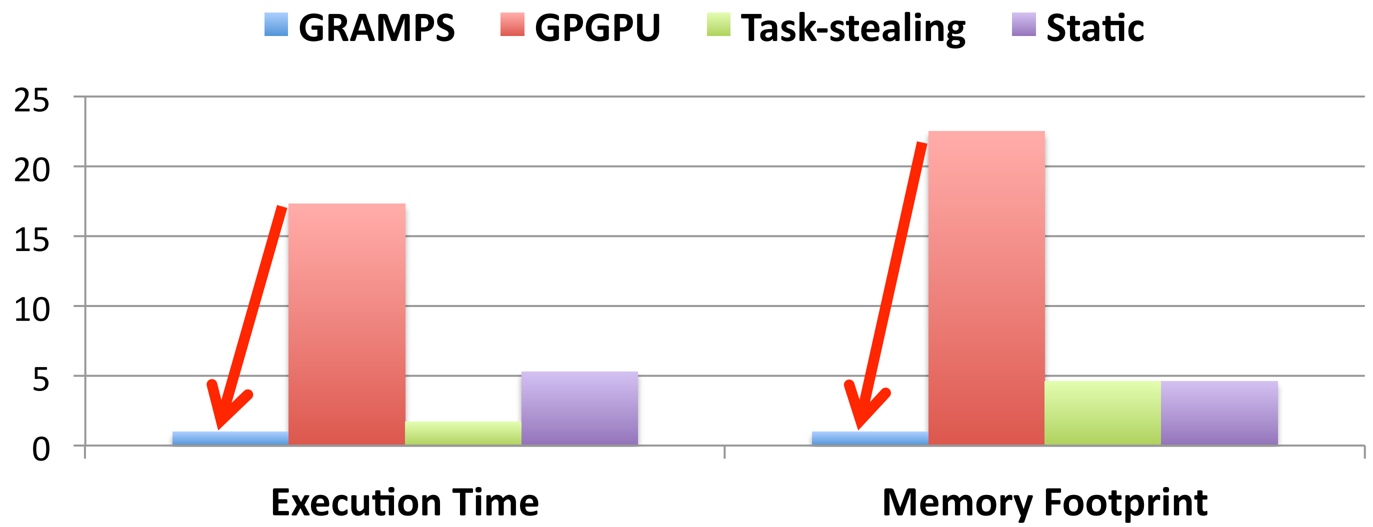
Locality-Aware Dynamic Scheduling





- GRAMPS scheduler [PACT'11]
  - Breadth-first stage ordering: higher priority to consumers
  - Dequeue from high-priority first: good locality, small footprint
  - Steal low-priority first: good locality, less stealing
  - Backpressure to control footprint: full queue → block producers

Locality-Aware Dynamic Scheduling



- On a 12-core, 24-thread x86 system
  - Perf gains: 17x over GPGPU, 2-5x over task-stealing & static
  - Energy gains: 22x over GPGPU, 5x over task-stealing & static
  - Differences become larger as we scale up

Locality-aware Parallelism: Challenges Ahead

- Locality-aware scheduling for unstructured codes
  - E.g., collision detection, hash-join, …
- Applications with low temporal locality
  - E.g., probabilistic inference, graph algorithms, …
- Scheduling overheads at large-scale
  - E.g., >1K cores with fine-grain parallelism
  - HW scheduling: efficiency Vs flexibility

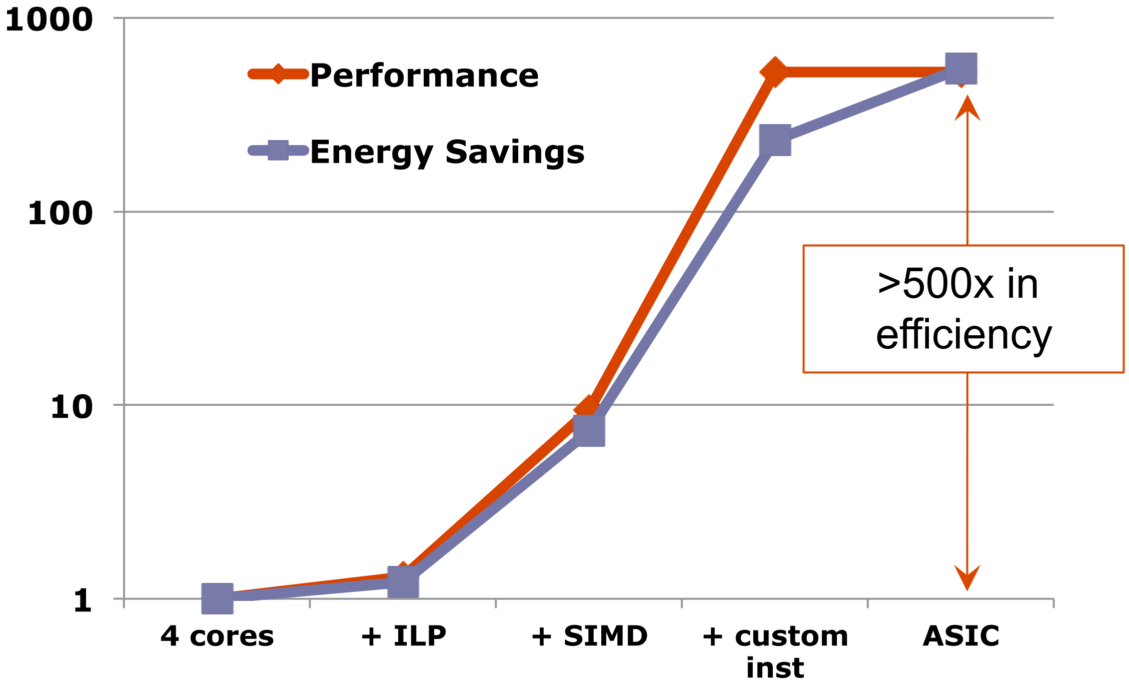
Advancing Systems without Technology Progress

- Locality-aware parallelism
- Specialization
- Reduce overprovisioning
- Increase utilization
- Approximate computing

Specialization

- Specialized cores are more efficient than general processor
  - In terms of energy and performance
- Already the norm in mobile chips
  - Now appearing on server chips as well
- Specialization flavors
  - Heterogeneous cores (e.g., big/little)
  - Specialized programmable cores (e.g., GPU, DSP)
  - Specialized functional units (e.g., SIMD, low-precision math)
  - Specialized fixed-function units (e.g., video and security engines)
  - Custom chip (ASIC)

Example: H.264 Transcoding [ISCA'10]

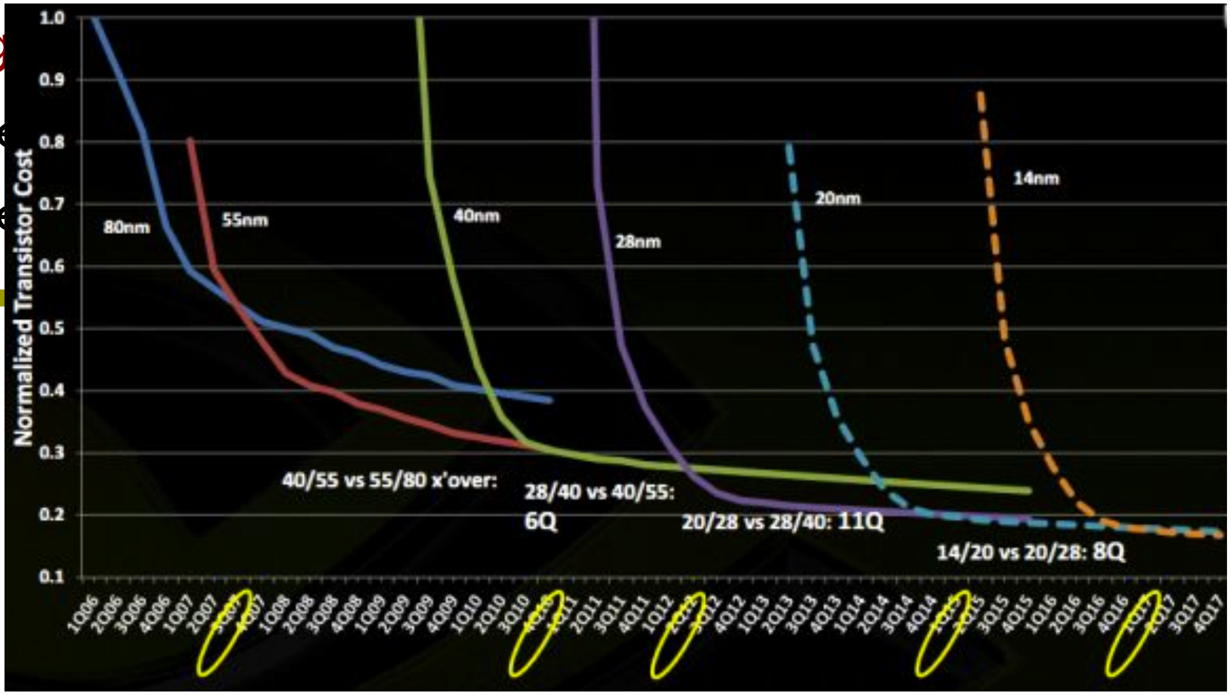


>500x in efficiency

- Lessons learned
  - High potential for efficiency gains
  - Multi-core or big/little cores are not enough
  - Amortize instruction overheads, maximize data reuse
  - Need extreme specialization & lots of area for highest gains

Specialization: Challenges Ahead

- The design problem
  - Need fast & efficient design/verification of specialized units
- The cost problem
  - Need to contain the cost of specialized units
- The g
  - Ne
  - Fle

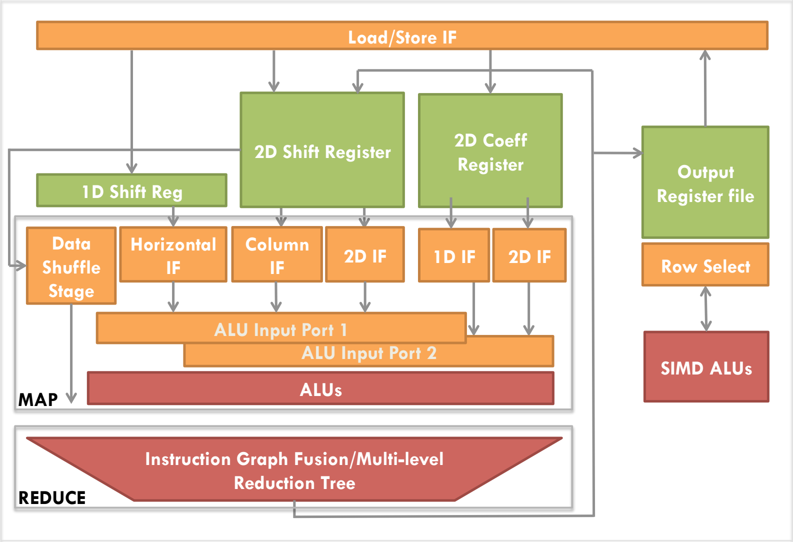


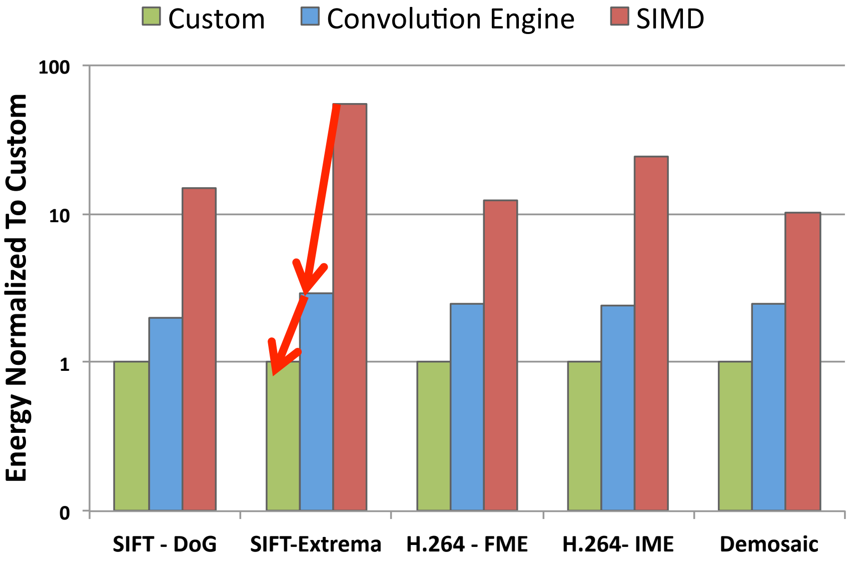
Balancing Efficiency & Flexibility

- Insight: domain-specific accelerators
  - Specialize to data-flow & locality of common patterns
  - Flexibility through configurable compute
- Example: convolutions in image, video, and vision processing
  - Common: convolutions as map-reduce computations over stencils
  - Differences: stencil dimensions and size, map and reduce operations

	Data-flow	Map	Reduce	Stencil
H.264 IME	2D convolution	Abs difference	Add	4x4
H.264 FME	1D convolution	Multiply	Add	6
SIFT DoG	2M matrix op	Subtract	-	
SIFT Extrema	1D convolution	Compare	And	3
Demosaic interpolation	1D convolution	Multiply	Graph fusion	3

Balancing Efficiency & Flexibility





- Convolution engine [ISCA'13]
  - Custom register files for different access patterns
  - Configurable 10b ALUs for different convolution types
- Efficiency: 2-3x worse area and energy of custom unit
  - But 8-15x better than SIMD engines, 100x better than multi-core

Advancing Systems without Technology Progress

- Locality-aware parallelism
- Specialization
- Reduce overprovisioning
- Increase utilization
- Approximate computing

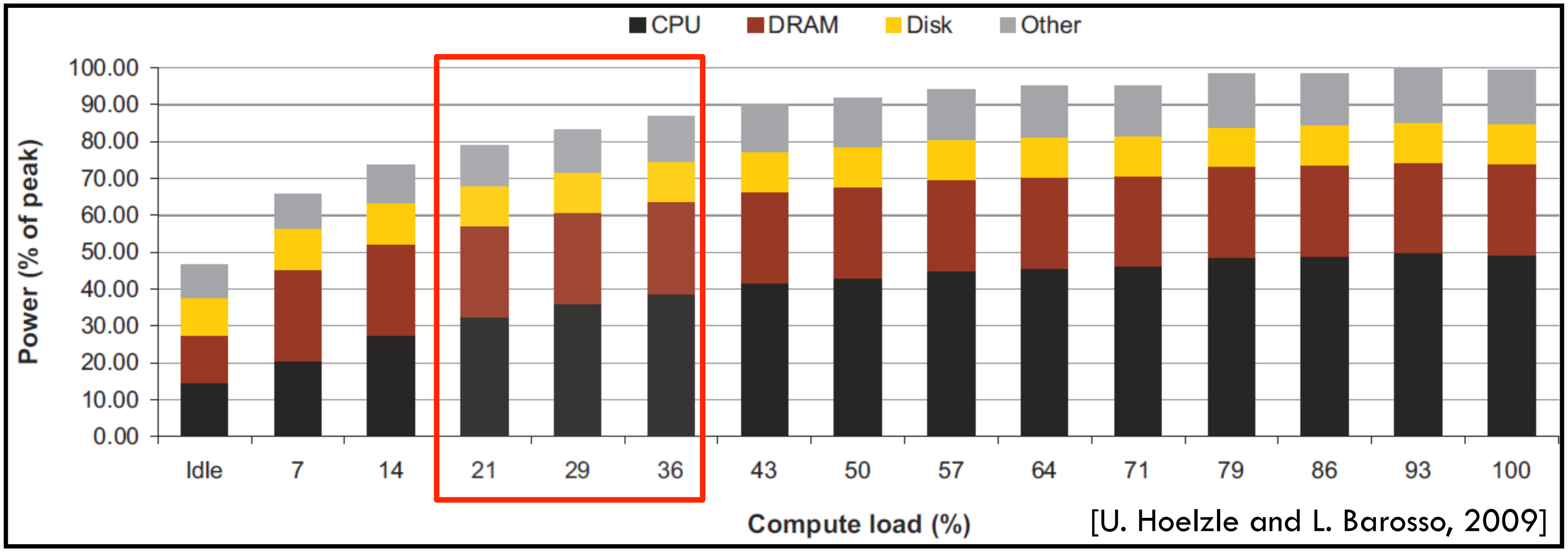
Software Bloat

[S. Amarasinghe, 2011]

PHP	9,298,440 ms	51,090x
Python	6,145,070 ms	33,764x
Java	348,749 ms	1816x
C	19,564 ms	107x
Tiled C	12,887 ms	71x
Vectorized	6,607 ms	36x
BLAS Parallel	182 ms	1

- Deep SW stacks needed for complex functionality
  - But few optimizations target cross-layer efficiencies
- Challenge: functionality & ease of use without efficiency loss

HW Bloat:
Main Memory Power in Datacenters



[U. Hoelzle and L. Barosso, 2009]

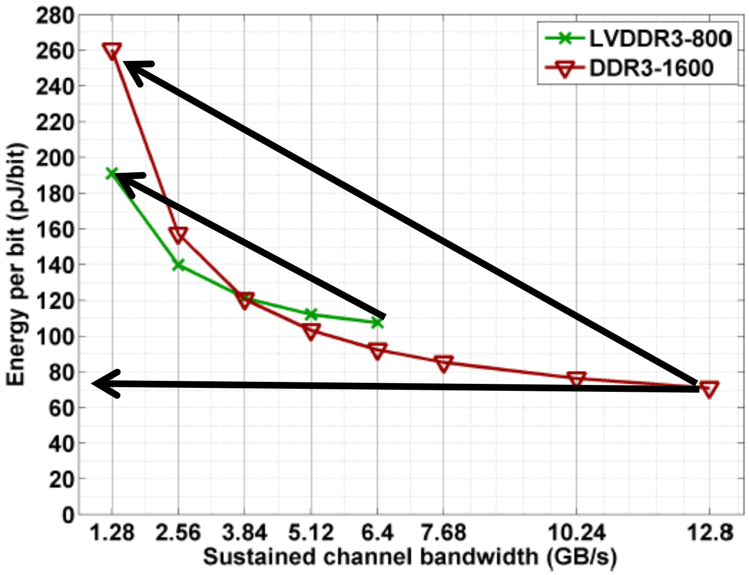
- Server power main energy bottleneck in datacenters
  - PUE of ~1.1 → the rest of the system is energy efficient
- Significant main memory (DRAM) power
  - 25-40% of server power across all utilization points

DDR3 Energy Characteristics

- DDR3 optimized for high bandwidth (1.5V, 800MHz)
  - On chip DLLs & on-die-termination lead to high static power
  - 70pJ/bit @ 100%, 260pJ/bit at 10%

- LVDDR3 alternative (1.35V, 400MHz)
  - Lower Vdd → higher on-die-termination
  - Still disproportional

- Need memory systems that consume lower energy and are proportional
  - What metric can we trade for efficiency?



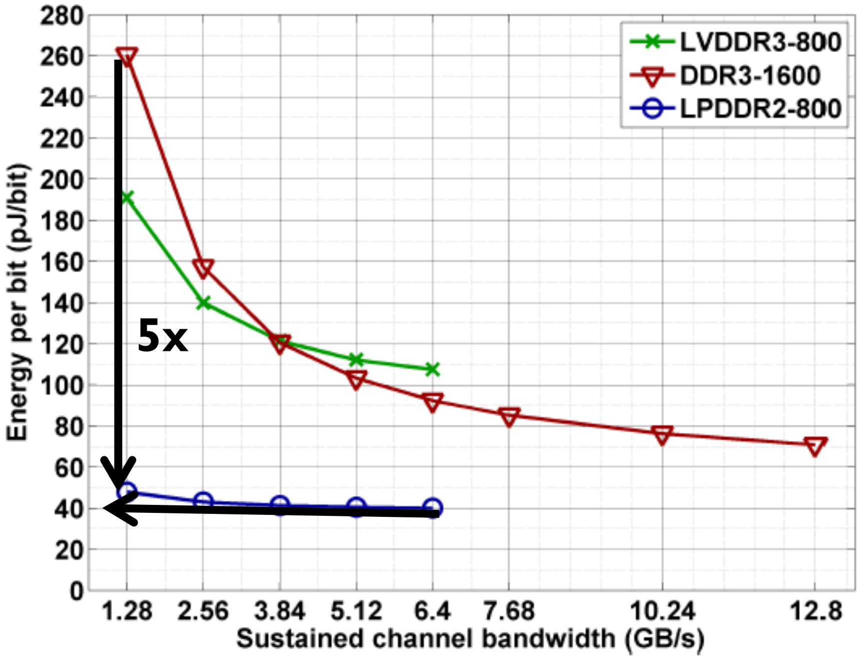
Memory Use in Datacenters

Resource Utilization for Microsoft Services under Stress Testing [Micro'11]

	CPU Utilization	**Memory BW Utilization**	Disk BW Utilization
Large-scale analytics	88%	**1.6%**	8%
Search	97%	**5.8%**	36%

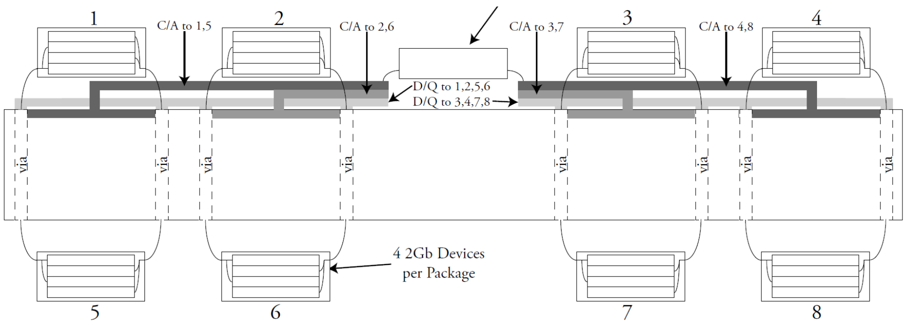
- Online apps rely on memory capacity, density, reliability
  - Web-search and map-reduce
    - CPU or DRAM latency bound
  - Memory caching, DRAM-based storage, social media
    - Bound by network bandwidth
- We can trade off bandwidth for energy efficiency

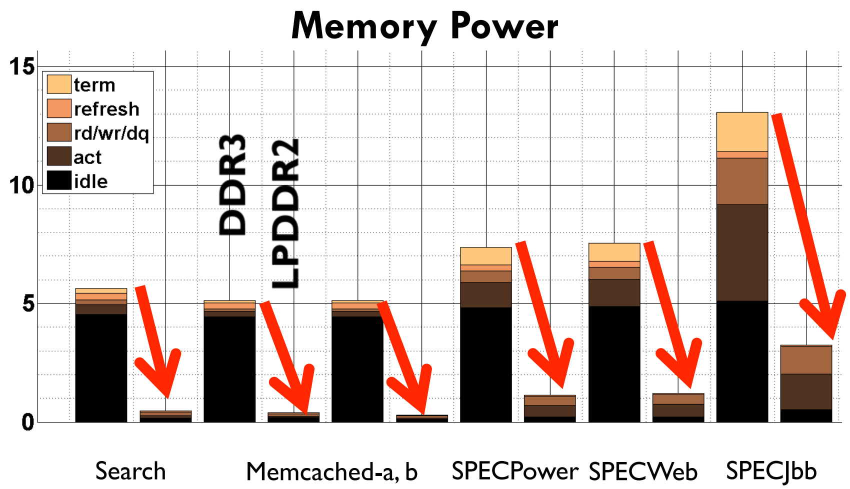
Mobile DRAMs for Datacenter Servers [ISCA'12]



- Similar core, capacity, and latency as DDR3
- Interface optimized for lower power & lower bandwidth ($^1/_2$)
  - No termination, lower frequency, faster power-down modes
- Energy proportional & energy efficient

Mobile DRAMs for Datacenter Servers [ISCA'12]





- LPDDR2 module: die stacking + buffered module design
  - High capacity + good signal integrity
- 5x reduction in memory power, no performance loss
  - Save power or increase capability in TCO neutral manner
- Unintended consequences
  - Energy efficient DRAM → L3 cache power now dominates

Reduce Overprovisioning: Challenges Ahead

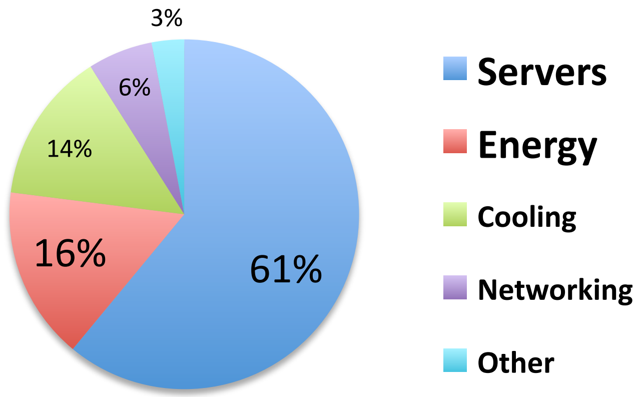
- Automatic cross-layer software optimizations
  - Static and dynamic
- End-to-end HW design for resource efficiency
  - From efficient components to efficient full-systems
- Linking resource efficiency and performance
  - Key requirement for management policies

Advancing Systems without Technology Progress

- Locality-aware parallelism
- Specialization
- Reduce overprovisioning
- Increase utilization
- Approximate computing

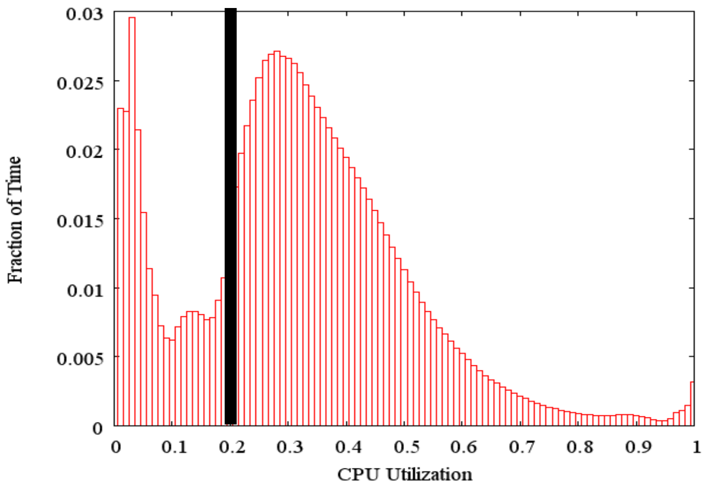
Server Utilization in Datacenters

Total Cost of Ownership



[J. Hamilton, http://mvdirona.com]

Server utilization



[U. Hoelzle and L. Barosso, 2009]

- Servers dominate datacenter cost
  - CapEx and OpEx
- Server resources are poorly utilized
  - CPUs cores, memory, storage

Low Utilization

- Primary reasons
  - Diurnal user traffic & unexpected spikes
  - Planning for future traffic growth
  - Difficulty of designing balanced servers
- Higher utilization through workload co-scheduling
  - Analytics run on front-end servers when traffic is low
  - Spiking services overflow on servers for other services
  - Servers with unused resources export them to other servers
- *So, why hasn't co-scheduling solved the problem yet?*

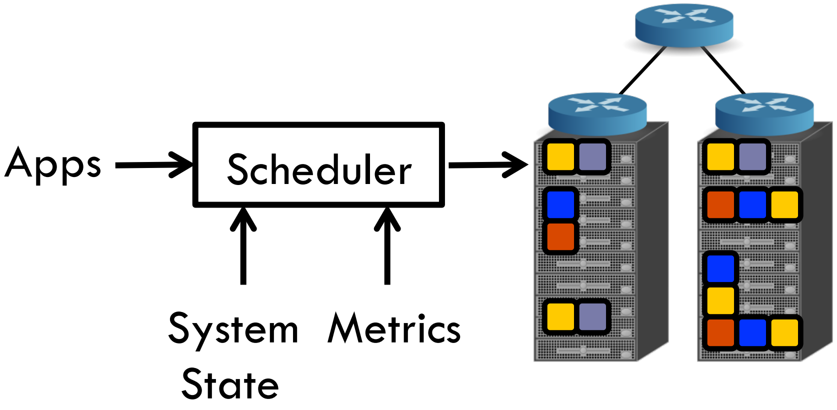
Interference→ Poor Performance & QoS

- Interference on shared resources
  - Cores, caches, memory, storage, network
  - Large performance losses (e.g., 40% for Google apps)

- QoS issue for latency-critical applications
  - Optimized for for low 99th percentile latency in addition to throughput
  - Small fraction of strugglers can lead to large QoS degradation

- Common cures lead to poor utilization
  - Limited resource sharing
  - Exaggerated reservations

Datacenter Scheduling

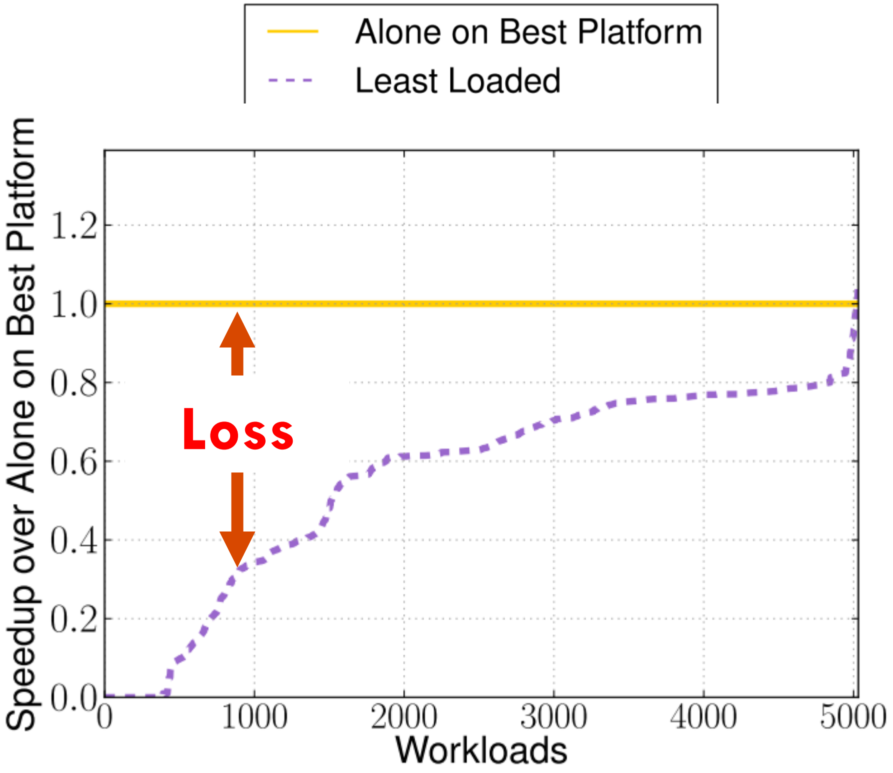


Apps

Scheduler

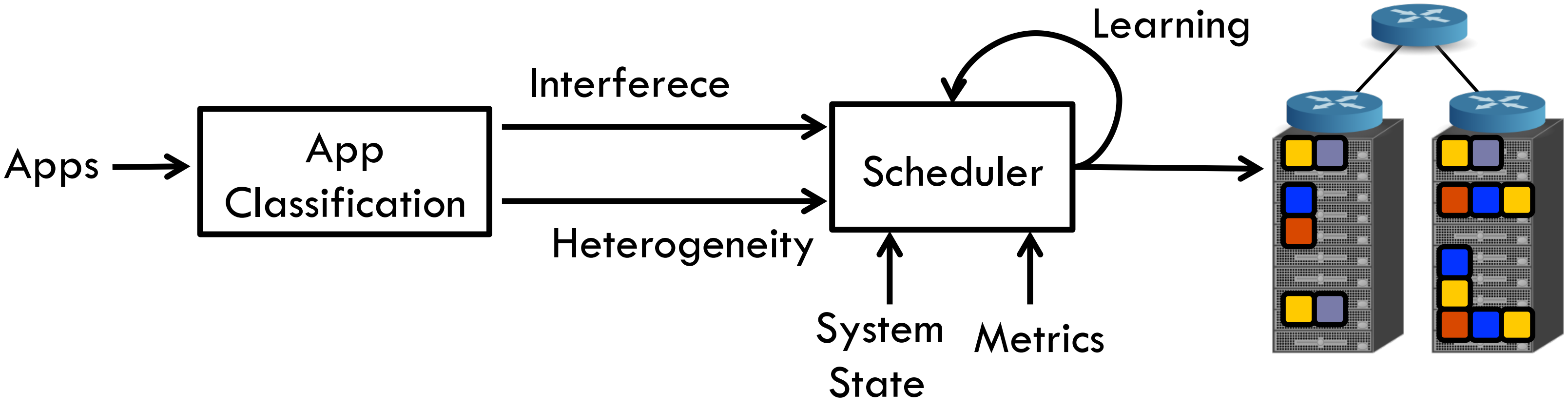
System State

Metrics



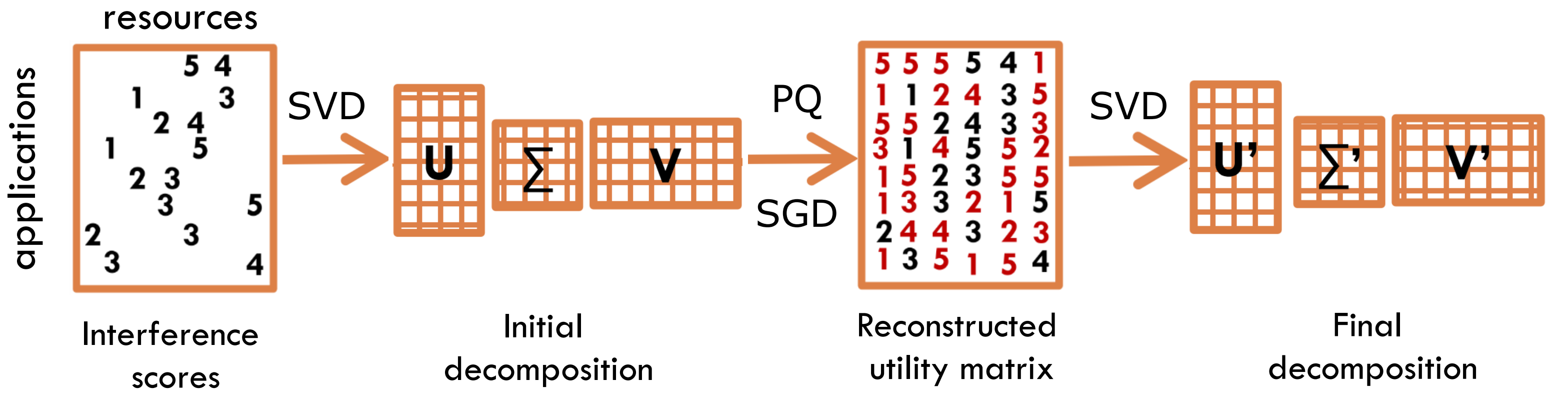
- Two obstacles to good performance
  - Interference: sharing resources with other apps
  - Heterogeneity: running on suboptimal server configuration

Interference-aware Scheduling [ASPLOS'13]



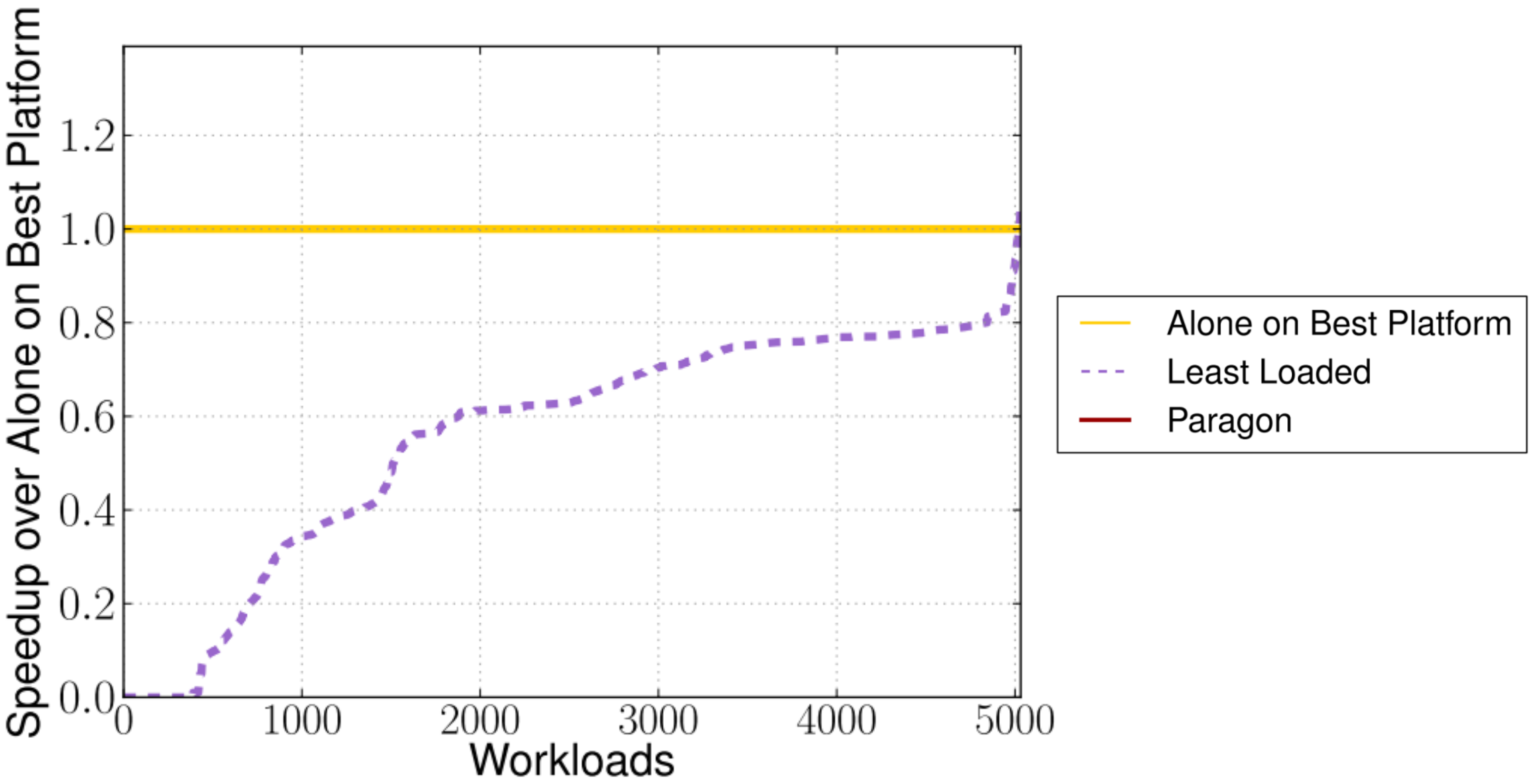
- Quickly classify incoming apps
  - For heterogeneity and interference caused/tolerated
- Heterogeneity & interference aware scheduling
  - Co-schedule apps that don't interfere much
  - Send apps to best possible server configuration
- Monitor & adapt
  - Deviation from expected behavior signals error or phase change

Fast & Accurate Classification



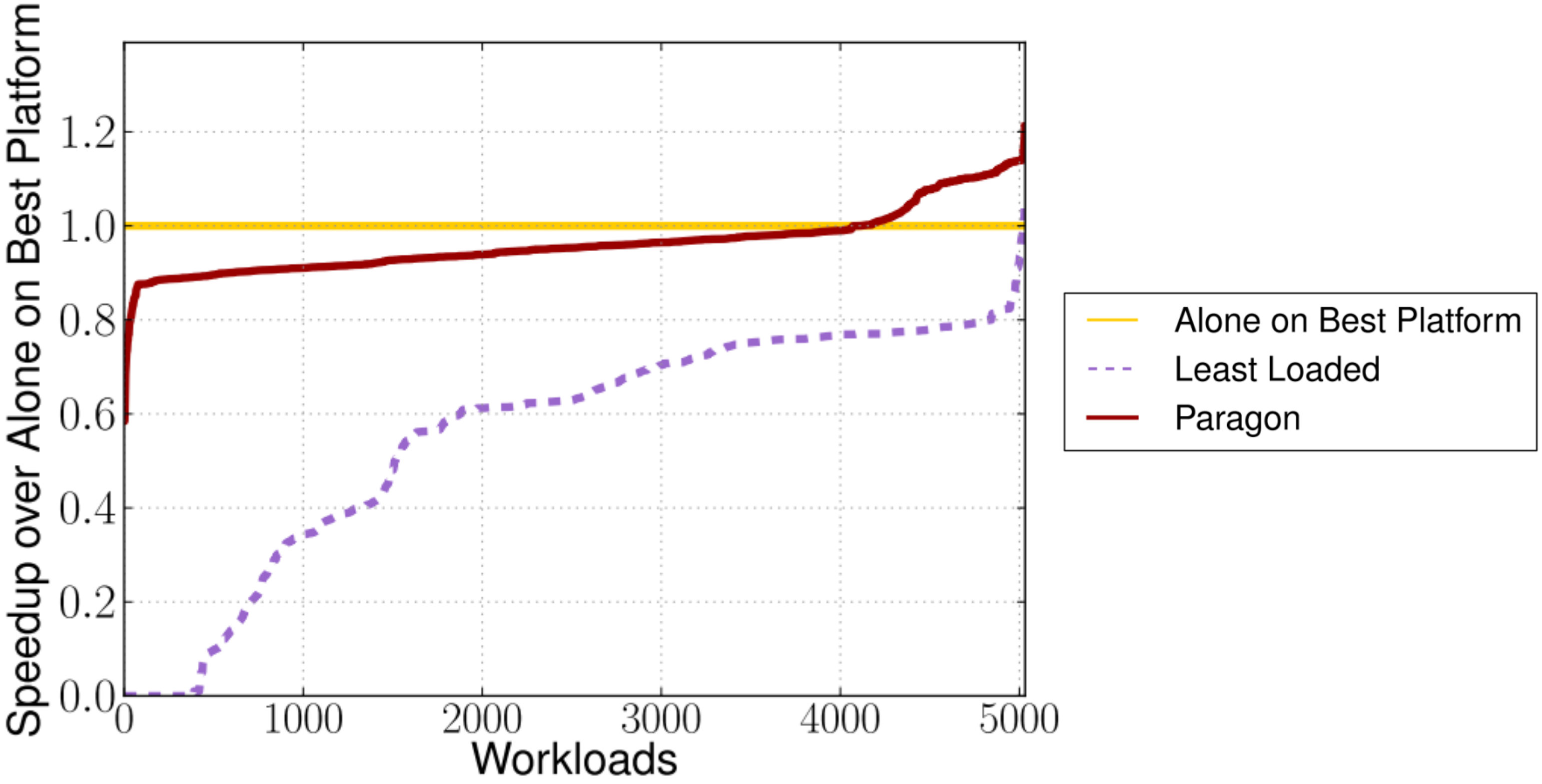
- Cannot afford to exhaustively analyze workloads
  - High churn rates of evolving and/or unknown apps
- Classification using collaborative filtering
  - Similar to recommendations for movies and e-commerce
  - Leverage knowledge from previously scheduled apps
  - Can classify accurately within 1 min from app arrival
    - For both interference and heterogeneity

Interference-aware Scheduling [ASPLOS'13]



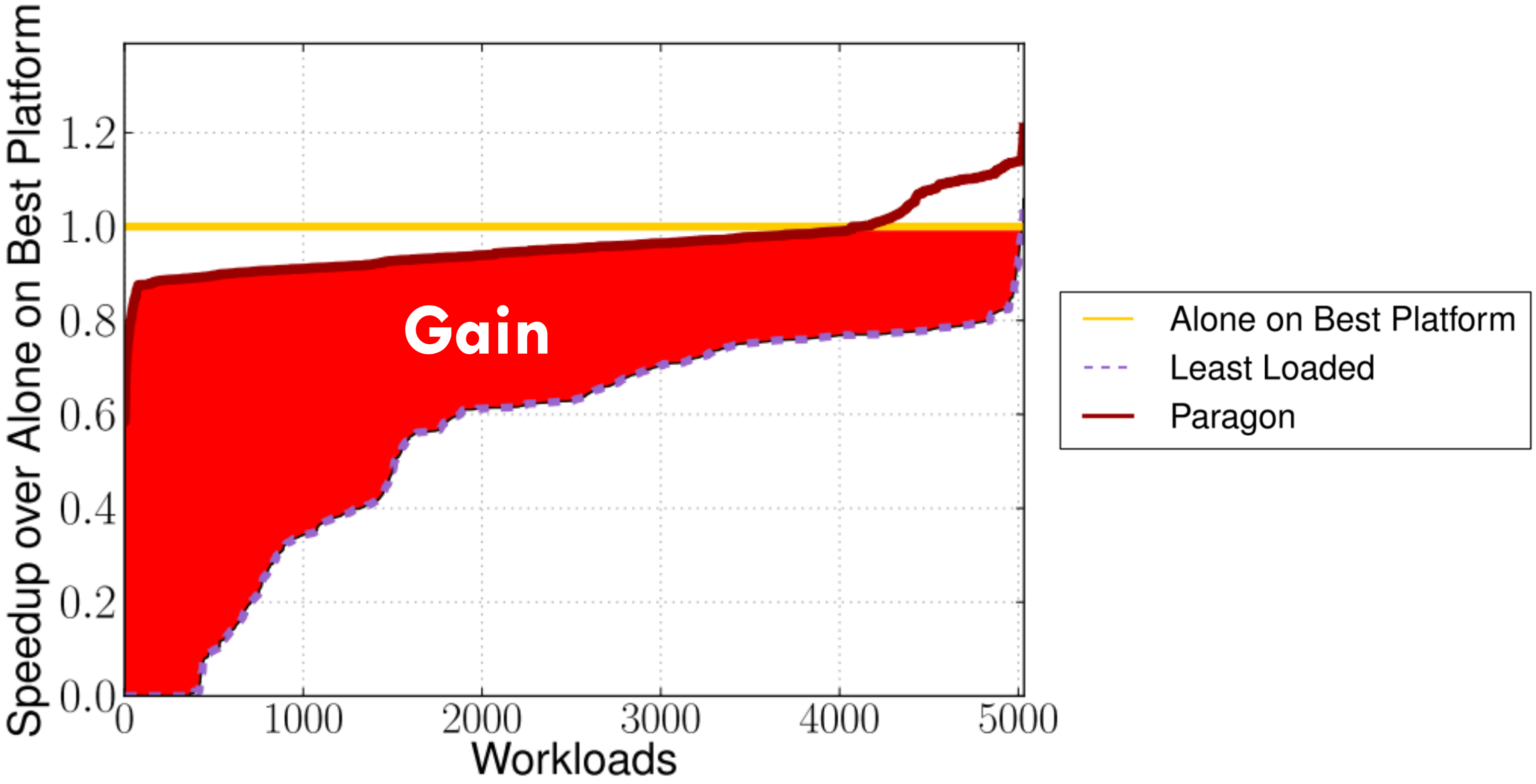
- 5K apps on 1K EC2 instances (14 server types)

Interference-aware Scheduling [ASPLOS'13]



- Better performance with same resources
  - Most workloads within 10% of ideal performance

Interference-aware Scheduling [ASPLOS'13]



- Better performance with same resources
  - Most workloads within 10% of ideal performance
  - Can serve additional apps without the need for more HW

Increase System Utilization: Challenges Ahead

- Isolation mechanisms
  - Mechanisms for fine-grain sharing and priorities
  - CPU, caches, memory, I/O
- Management policies
  - Latency-critical Vs. batch apps
  - Static Vs. dynamic, local Vs. global policies
  - Interactions with provisioning and pricing models
- Implications for application development

Advancing Systems without Technology Progress

- Locality-aware parallelism
- Specialization
- Reduce overprovisioning
- Increase system utilization
- Approximate computing

Approximate Computing

- Now: high-precision outputs from deterministic HW
  - Requires high operation count in software
  - Requires high margins in hardware
- Approximate outputs are often sufficient
  - Machine learning, computer vision, search, physical simulation, …
- Projected benefits: from 20% to 200x
  - Performance and/or energy

Taxonomy of Approximation Techniques

- Inexact SW on exact HW
  - E.g., code perforation, Green
- Exact SW on inexact HW
  - E.g., Razor
- Inexact SW on inexact HW
  - E.g., probabilistic processors, Truffle
- Approximation with neural networks
  - E.g., IBM SyNAPSE, NPUs
- Analog accelerators for approximation
  - E.g., Intel ETANN, Inria NPU

Approximate Computing: Challenges Ahead

- End-to-end management of errors
  - From user goals to HW and SW management
  - What is the best way to spend the error margin?
- Tools
  - Programming languages, compilers, runtime systems
- Digital Vs mixed-signal hardware
  - Efficiency, practicality, robustness

Summary

- CMOS scaling is over & new technologies are not ready

- Potentially 2-3 decades of scaling using
  - Locality-aware parallelism
  - Specialization
  - Reduce overprovisioning
  - Increase utilization
  - Approximate computing

- Cross-cutting research
  - Must revisit both sides of the HW/SW interface
  - Must revisit both chip and system level architecture

Acknowledgements

- Mark Hill (U. of Wisconsin)
    - Co-organizer of ISAT study in March 2012
- ISAT study participants
    - 48 from academia and industry
- MAST & VLSI groups at Stanford