

BLUE WATERS

SUSTAINED PETASCALE COMPUTING

Energy-aware Software Development for Massive-Scale Systems

Torsten Hoefler

With input from Marc Snir, Bill Gropp and Wen-mei Hwu

Colloquium talk at the University of Utah



GREAT LAKES CONSORTIUM
FOR PETASCALE COMPUTATION

Outline

- The HPC Energy Crisis
- Computer Architecture Speculations
- Algorithmic Power Estimates
- Network Power Consumption
- Power-aware Programming
 - Quick Primer on Power Modeling
- This is not an Exascale talk! But it's fun to look at!
- All images used in this talk belong to the owner!



Some Ammunition for Politics

- US EPA Report to Congress on Server and Data Center Energy Efficiency, Public Law 109-431
 - Data centers consumed 61 billion kilowatt-hours (kWh) in 2006 (1.5% of total U.S. electricity consumption)
 - Electricity cost of **\$4.5 billion** (~15 power plants)
 - **Doubled** from 2000-2006

*The New York Times**

July 31, 2011

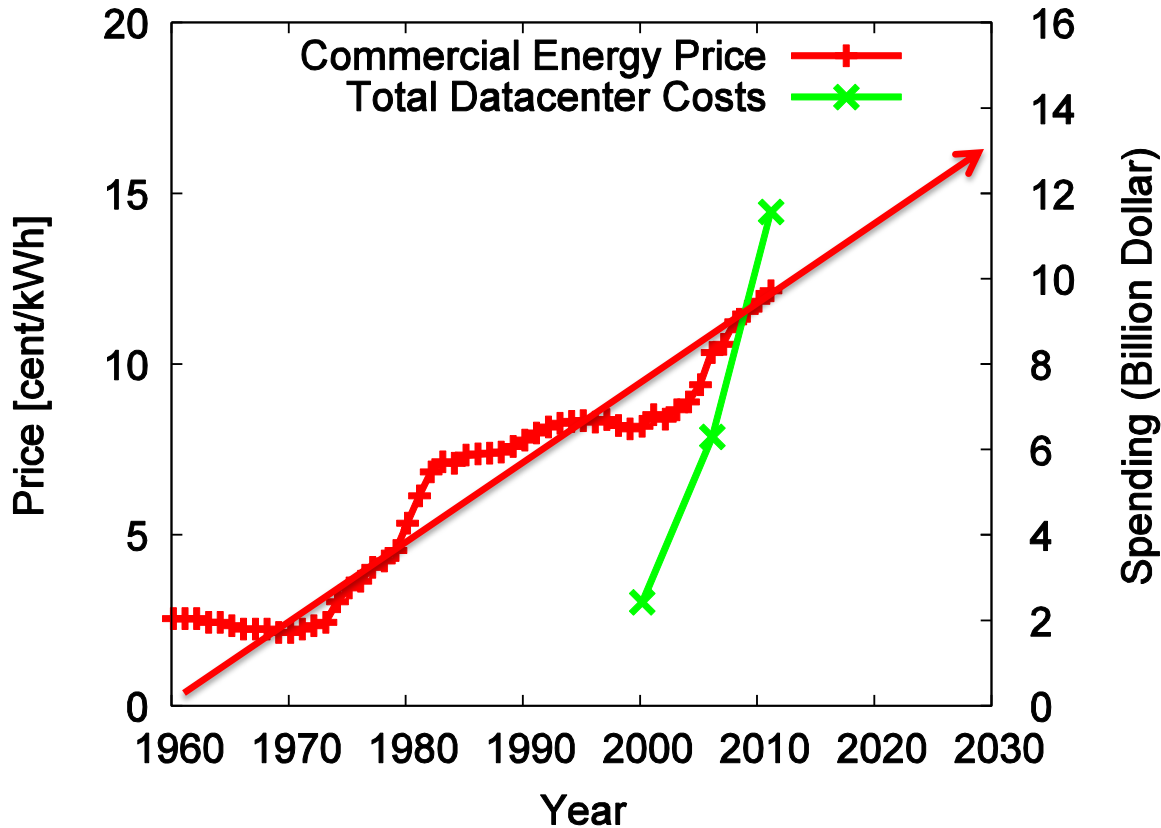
Data Centers' Power Use Less Than Was Expected

By JOHN MARKOFF

SAN FRANCISCO — Data centers' unquenchable thirst for electricity has been slaked by the global recession and

- Koomey's report (Jul. 2011)
 - Only **56% increase** through 2006-2011 though
 - Attributed to virtualization and economic crisis in 2008
 - Well, we're still on an **exponential** curve!

Development and Projection of Energy Costs



- Exponential requirements times linear cost growth: ☹️

Source: T. Hoefler: Software and Hardware Techniques for Power-Efficient HPC Networking

What is this “Energy Crisis”? (Short Version)

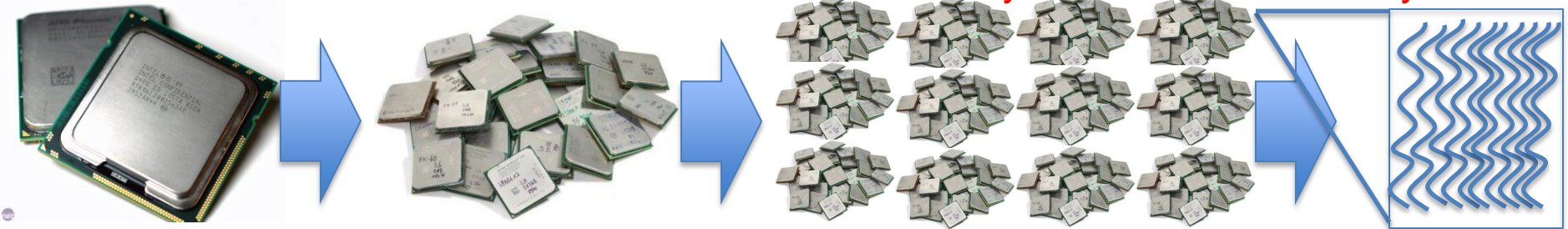
- Expectation: double performance every 18 months at roughly equal costs (including energy)
- Realization: **Explicit** parallelism at **all** levels
 - Instruction (massive out-of-order may end, ILP still important)
 - Memory (implicit caching and HW prefetch?)
 - Thread (simple tasking may not be efficient)
 - Process (separated address space overheads unaffordable?)

SMP

MPP

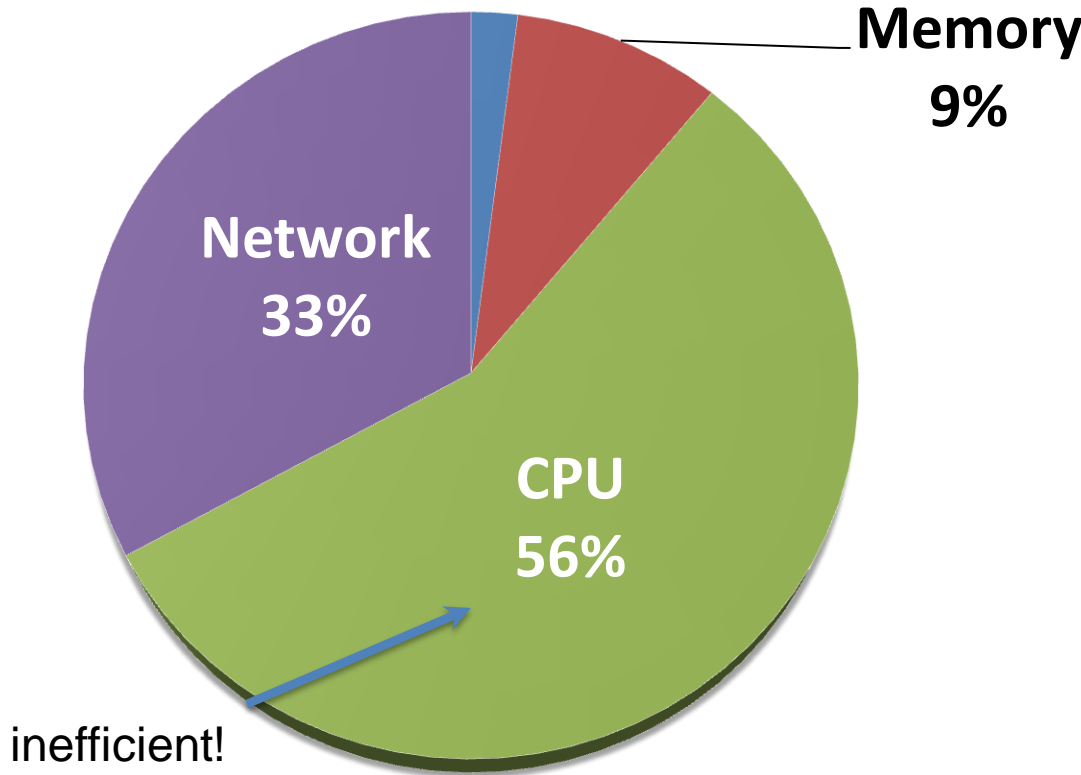
Many Core

Many Thread



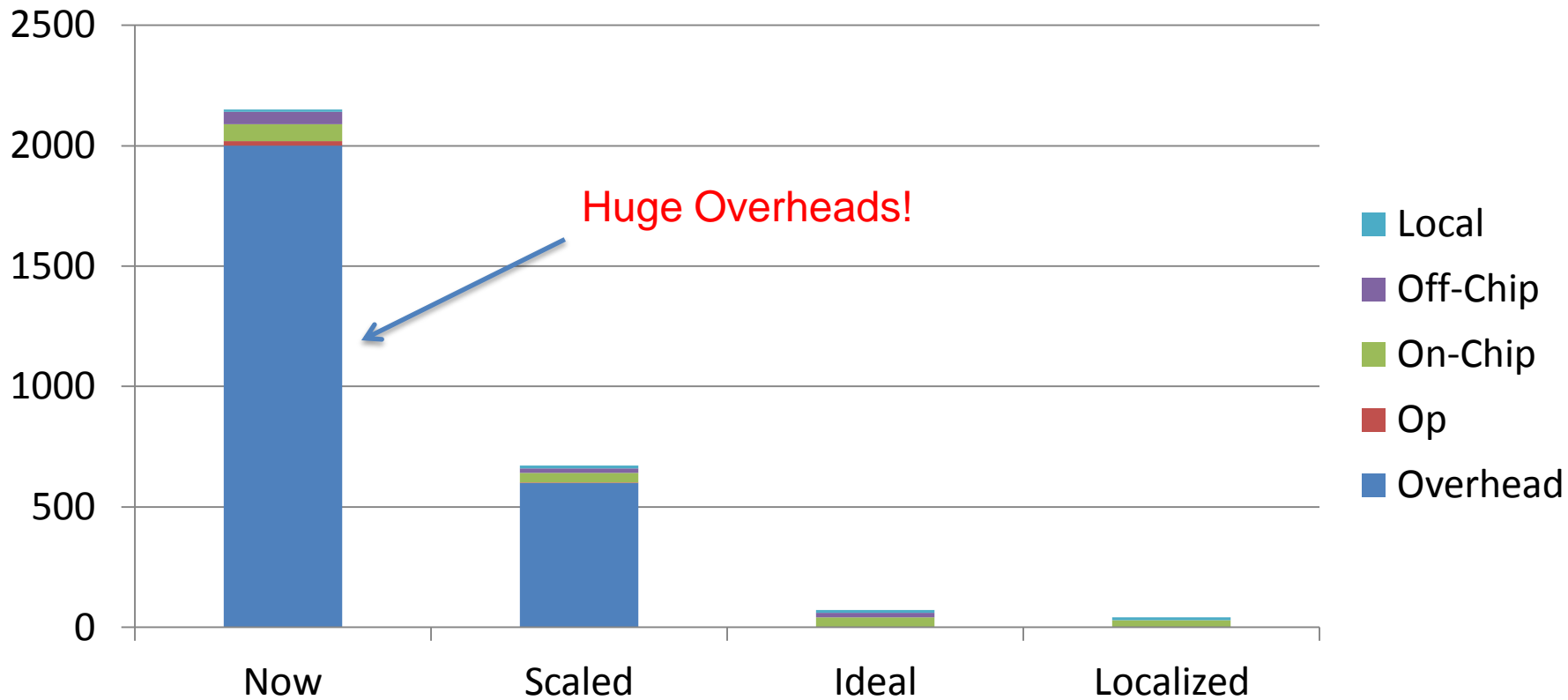
- Not only parallelism! → more parallelism!

System Power Breakdown Today (Longer Story)



Source: Kogge et al. Exascale Computing Study

CPU Power Consumption Prediction (56%)



- Overhead: Branch prediction, reg. renaming, spec. execution, OOO, decoding (x86), caches ...

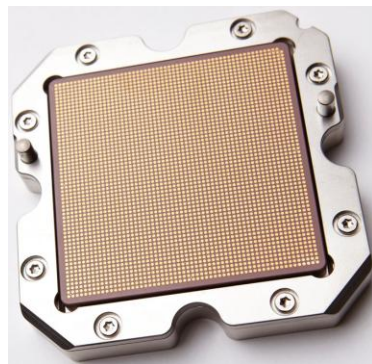
Source: Bill Dally, 2011

Current Commodity Architectural Solutions



Commodity

Superscalar
OOO issue
High power
Low perf.
Very cheap



Server

Superscalar
OOO issue
VLIW/EPIC?
Med. power
High perf.
Expensive



Vector

Low power
High perf.
Expensive

Vector pipe
Many registers
Pipelined mem.



GPGPU

Multi-threaded
Shared units
Parallel memory
Low power
Cheap



“Cell phone”

Many core
Specialized
Very Low power
Very Cheap

Future Power-aware Architectures?

- Overheads are too large!
 - Especially complex logic inside the CPU
 - Too complex instruction decode (esp. x86)
 - Excessive copying in OOO
- Architectures are simplified
 - E.g., Cell, SCC
 - Small or no OOO fetch and instruction window
 - Emphasize vector operations
 - Fix as much as possible during compile time
 - VLIW/EPIC comeback?

**BACK
TO
THE FUTURE**

(V)LIW/EPIC to the Rescue?

- (Very) Large Instruction Word ((V)LIW)
 - No dynamic operation scheduling (i.e., Superscalar)
 - Static scheduling, simple decode logic
- Explicit Parallel Instruction Computing (EPIC)
 - Groups of operations (bundles)
 - Stop bit indicates if bundle depends on previous bundles
- Complexity moved to compiler
 - Very popular in low-power devices (AMD/ATI GPUs)
 - But non-deterministic memory/cache times make static scheduling hard!

Trends in Algorithms (Towards Co-Design)

- Most early HPC applications used regular grids
 - Simple implementation and execution, structured
 - However, often not efficient
 - Needs to compute all grid points at full precision
- Adaptive Methods
 - Less FLOPs, more science!
 - Semi-structured
- Data-driven Methods
 - “Informatics” applications
 - Completely unstructured



The Full Spectrum of Algorithms

Structured

```
for (int i=0; i<N, i++)
  C[i] = A[i] + B[i]
```

VEC

MT

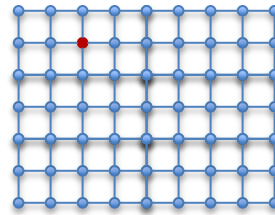
```
for (int i=0; i<N, i+=s)
  vec_add(A[i], B[i], C[i])
```

VLIW

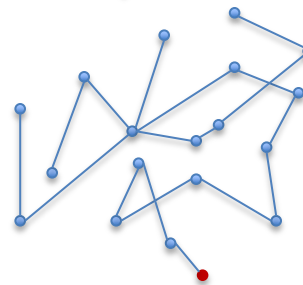
INT FP FP FP FF FP FP FP BR

```
for (int i=0; i<N, i++)
  spawn(A[i] = B[i]+C[i])
```

Algorithmic Trends



Less
Regular



Unstructured

```
while(v = Q.pop()) {
  for(int i=0, i<v.enum(), i++) {
    u = v.edges[i]; // mark u
    Q.push(u);
  }
}
```

VEC

MT

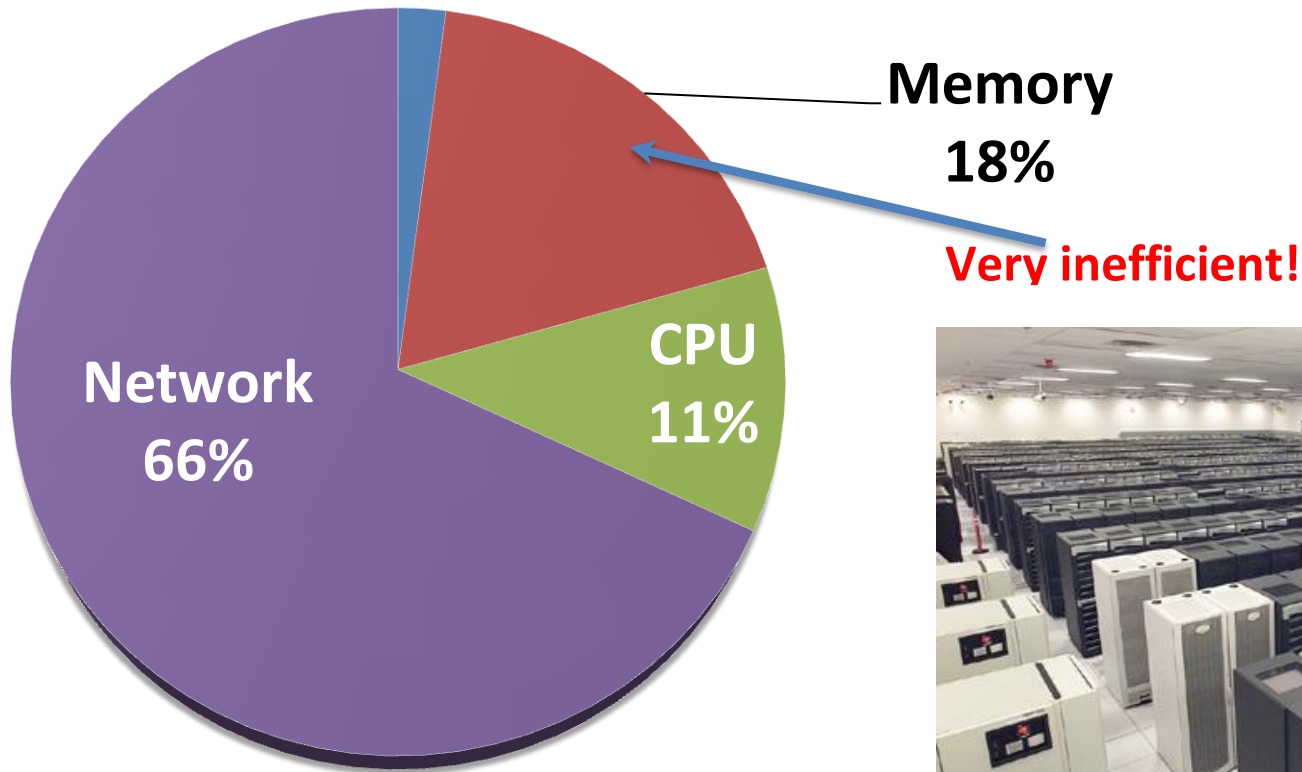
```
while(v = Q.pop()) {
  for(int i=0, i<v.enum(), i+=s) {
    vec_load(u, v.edges[i];
    vec_store(Q.end(), u);
  }
}
```

```
while(spawn(Q.pop())) {
  for(int i=0, i<v.enum(), i+=s) {
    spawn(update(v.edges[i], Q)
  }
}
```

General Architectural Observations

- Superscalar, RISC, wide OOO outside of power budget
 - Maybe “small/simple” versions
- VLIW/EPIC and Vector: very power-efficient
 - Performs best for static applications (e.g., graphics)
 - Problems with scheduling memory accesses
 - Limited performance for irregular applications with complex dependencies
- Multithreaded: versatile and efficient
 - Simple logic, low overhead for thread state
 - Good for irregular applications/complex dependencies
 - Fast synchronization (full/empty bits etc.)

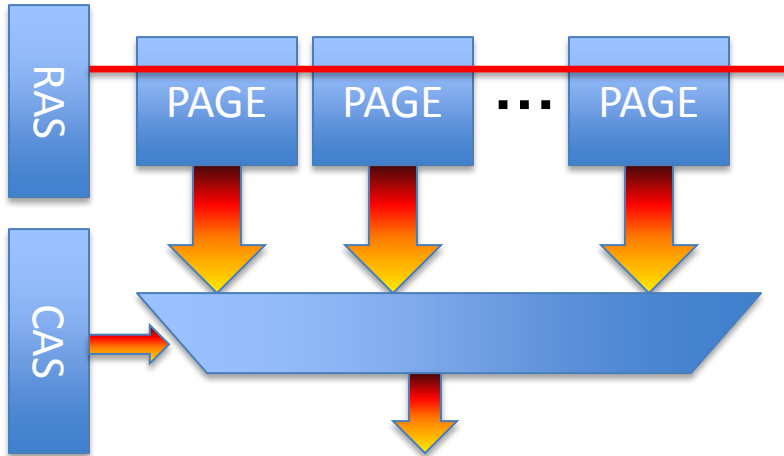
Optimized CPU System Power Consumption



Memory Power Consumption Prediction

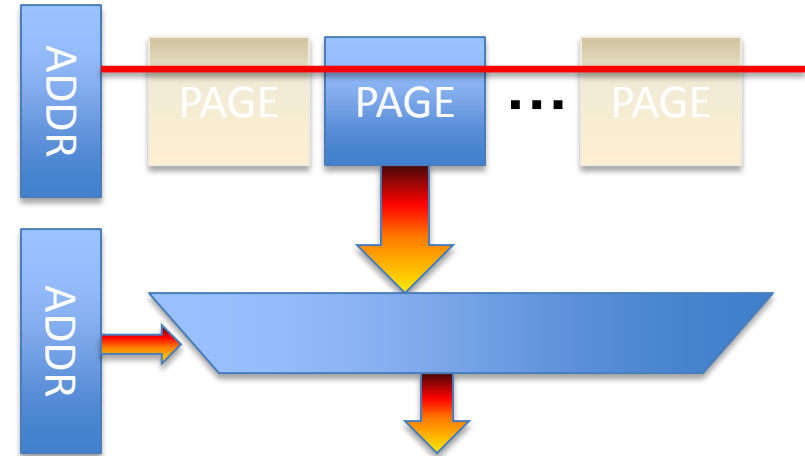
- DRAM Architecture (today ~ 2 nJ / 64 bit)

Current RAS/CAS-based



- All pages active
- Many refresh cycles
- Small part of read data is used
- Small number of pins

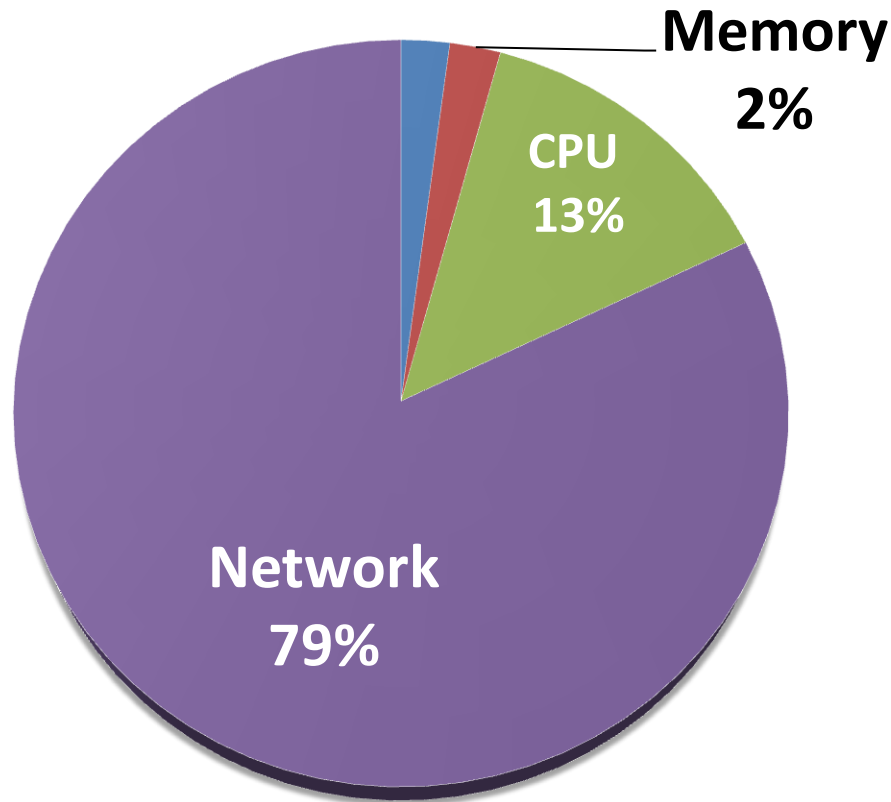
Desired Address-based



- Few pages active
- Read (refresh) only needed data
- All read data is used
- Large number of pins

- Cache/prefetch can be very wasteful \rightarrow scratchpad memory!

Optimized DRAM System Power Consumption

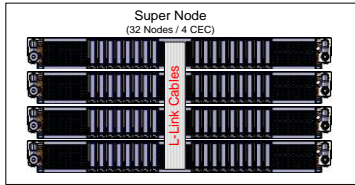


“The Network is the Computer”

- We must obey the network
 - Everything is a (hierarchical) network!

Building Block

SuperNode
(1024 cores)



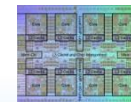
Drawer
(256 cores)



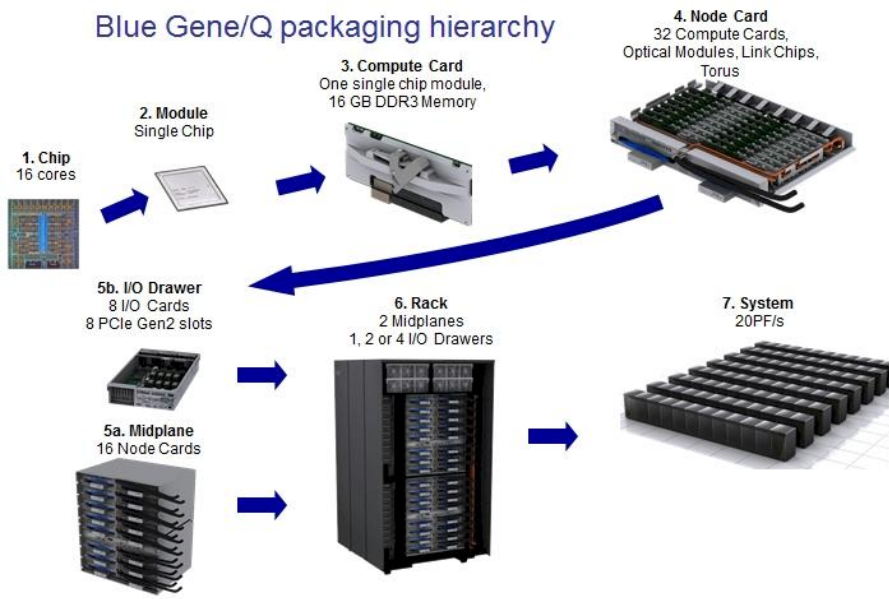
SMP node
(32 cores)



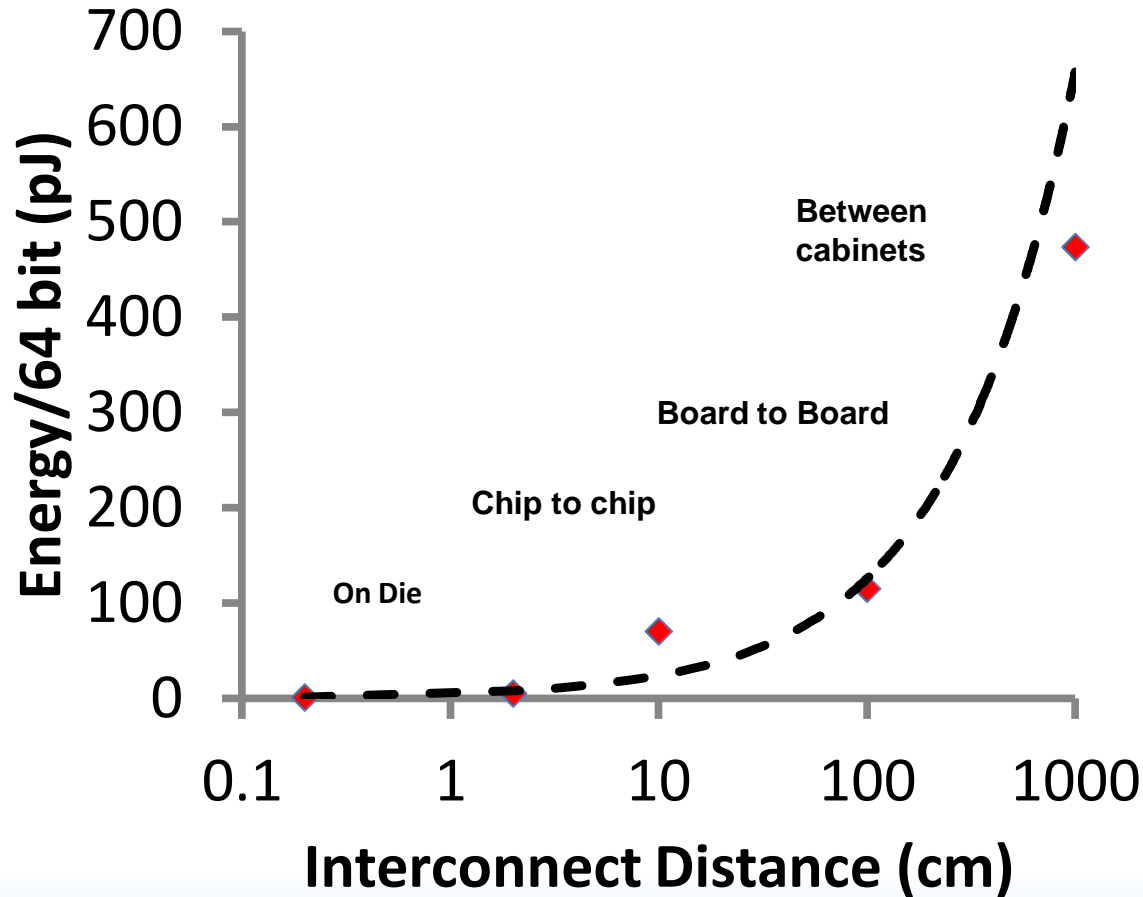
P7 Chip
(8 cores)



Blue Gene/Q packaging hierarchy



Network Power Consumption



Source: S. Borkar, Hot Interconnects 2011

A Quick Glance at Exascale

	Power	Scale
Exaflop	20 MW	Data Center
Petaflop	20 kW	Rack/Cabinet
Teraflop	20 W	Chip

- 20 MW → 20 pJ/Flop
 - 20% leakage → 16 pJ/Flop left
 - 7nm prediction: FPU needs 10 pJ/Flop
 - 6 pJ/Flop left for data movement 😊
 - Expected to be 10x-100x more!

ExaScale Computing Study:
Technology Challenges in
Achieving Exascale Systems

Peter Kogge, Editor & Study Lead
Keren Bergman
Shikhar Borkar
Dan Campbell
William Carlson
William Dally
Marty Dronson
Paul Franconi
William Harrod
Kerry Hill
Jon Hiller
Srivatsan Karp
Stephen Keckler
Dean Kretz
Robert Lucas
Mark Richards
AJ Scarpelli
Steven Scott
Alan Saville
Thomas Sterling
K. Stanley Williams
Katherine Yelick

September 28, 2008

This work was sponsored by DARPA IPTO in the Exascale Computing Study with Dr. William Harrod as Program Manager, AFRL contract number FA9550-07-4-0024. This report is published in the interest of scientific and technical information exchange and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

NOTICE

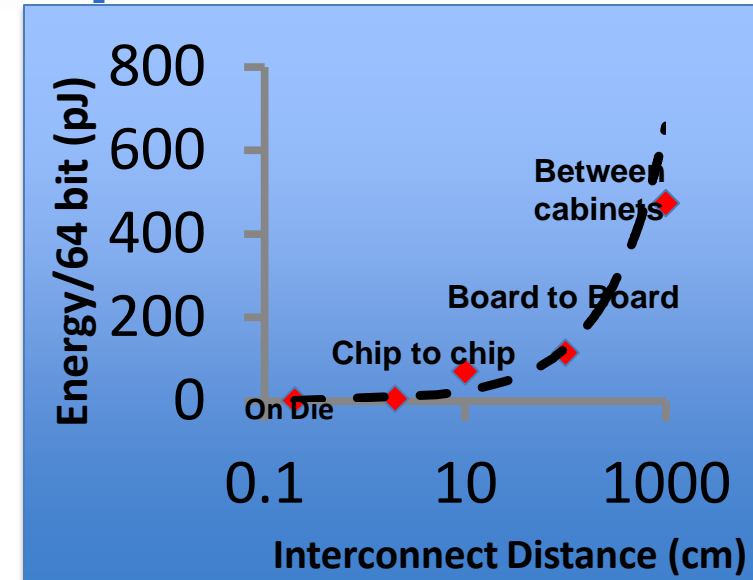
Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government furnished or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation, or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.



Programming a “Network Computer”

- Surprise: Locality is important!
 - Energy consumption grows with distance
- “Hidden” distribution: OpenMP
 - Problem: locality not exposed
- “Explicit” distribution: PGAS, MPI
 - User handles locality
 - MPI supports process mapping
- Probably MPI+X in the future



But what is

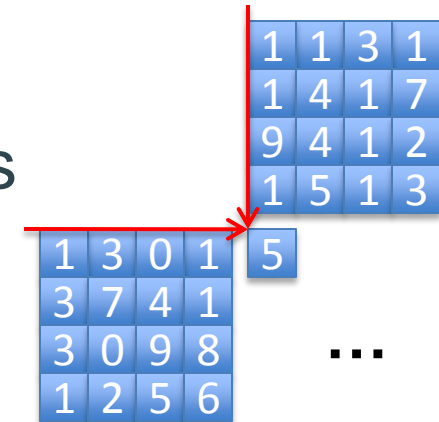


So, is it really about Flops? Of course not!

- But: Flops is the default algorithm measure
 - Often set equal to algorithmic (time) complexity
 - Numerous papers to reduce number of Flops
 - Merriam Webster: “flop: to fail completely”
- HPC is power-limited!
 - Flops are cheap, data movement is expensive, right?
- Just like using the DRAM architecture from the 80’s, we use algorithmic techniques from the 70’s!
 - **Need to consider I/O complexity instead of FLOPS**
 - Good place to start reading: Hong&Kung: *Red-Blue Pebble Game*

How much Data Movement is Needed? MatMul?

- Matrix Multiplication: $A=BC$
 - $N \times N$ matrix, $\geq 2N^2$ reads, $\geq N^2$ writes
- Textbook algorithm has no reuse
 - Example memory hierarchy model:

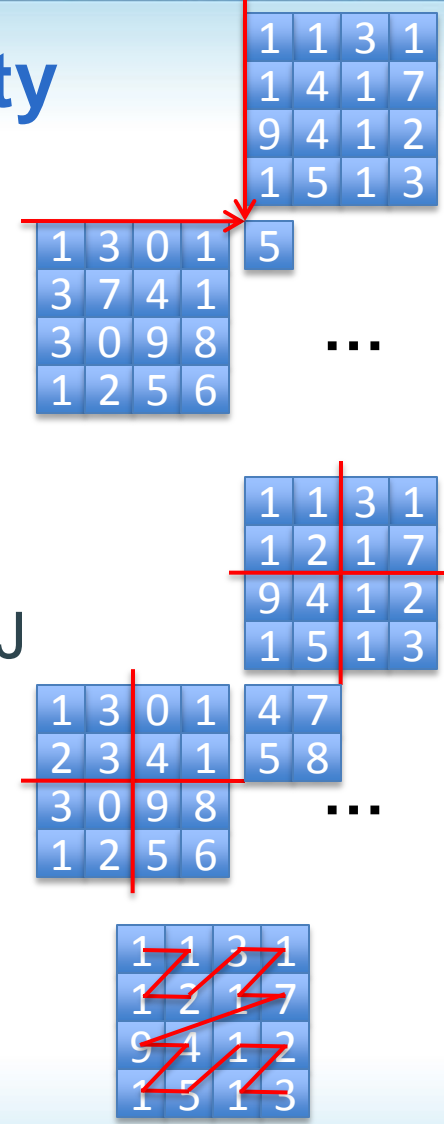


Functionality	Energy	Performance	Capacity (FP)
Core/FP Unit	50 pJ	125 ps	-
Register Bank	10 pJ	250 ps	100
Cache/SRAM	100 pJ	2 ns	100.000
Memory/DRAM	1000 pJ	100 ns	100.000.000

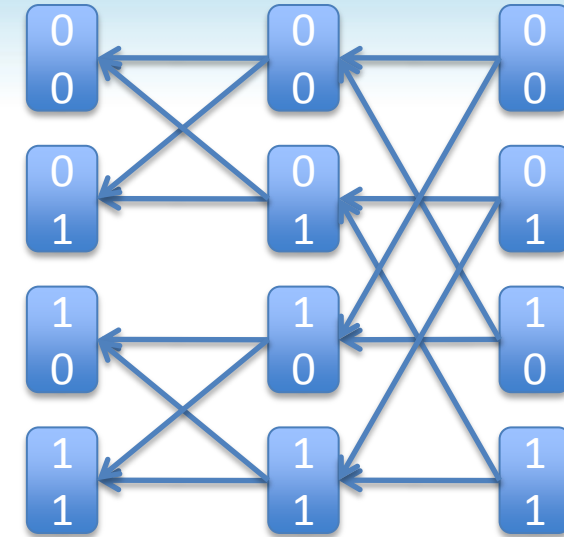
Source: Dally, 2011

I/O Complexity and Power Complexity

- Trivial algorithm (no reuse):
 - $E(N) = (2N^3 + N^2) * 1 \text{ nJ}$
 - $E(55\text{k}) = 332.75 \text{ kJ}$
 - $FP(55\text{k}) = 55.000^3 * 50 \text{ pJ} = 8.32 \text{ kJ}$
- Block algorithm (CxC blocks fit in cache)
 - $E(N,C) = [\text{DRAM ops}] * 1 \text{ nJ} + [\text{Cache ops}] * 0.1 \text{ nJ}$
 - $E(55\text{k},35) = 10.78 \text{ kJ} + 21.48 \text{ kJ} = 32.26 \text{ kJ}$
 - Can be improved with space-filling curves
- Memory locality makes algorithm feasible!
 - However, MM is extremely structured!
 - See Kogge's HPL study for another example!



Fast Fourier Transform



- N point transform (lower bounds!!)
 - $5N \log N$ FP operations
 - Cache of size $C + R$ registers
 - I/O lower bound (Hong&Kung):
 - $E(N) = (N \log N / \log C) + (N \log N / \log R) * 0.1 + (N \log N) * 0.01$ [nJ]
 - $FP(N) = 5N \log N * 50$ pJ
 - $E(100M) = 0.22$ J (2.65 J w/o cache) | $FP(100M) = 0.66$ J
 - $E(100G) = 300$ J (3.65 kJ w/o cache) | $FP(100G) = 913$ J
 - Caches are well-dimensioned
 - Hiding access costs, FP costs dominate (depending on constants)
 - Model can easily be extended to remote communication

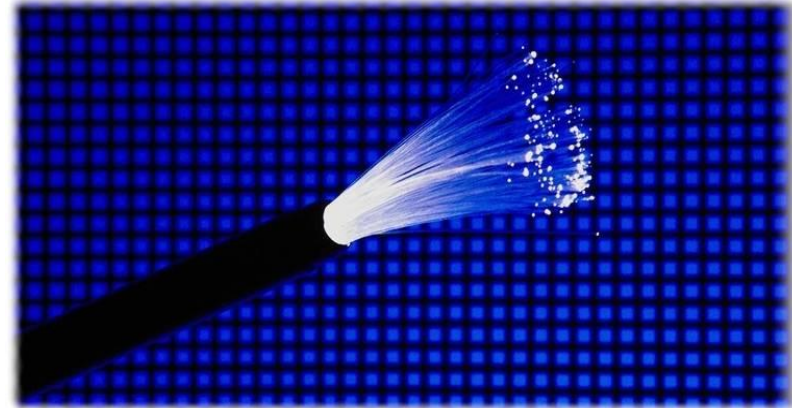
Power Consumption of Traditional Networks

- Most networks draw constant power
 - Full speed link protocol
- Some networks (will) have innovative features
 - E.g., InfiniBand's dynamic throttling
 - Potential problems: “network noise”? [Hoefler et al.'09]
- Other power-saving options
 - Network power states (explicit throttling)
 - Power-aware routing (source vs. distributed routing)
 - Application-specific routing (“compiled”)

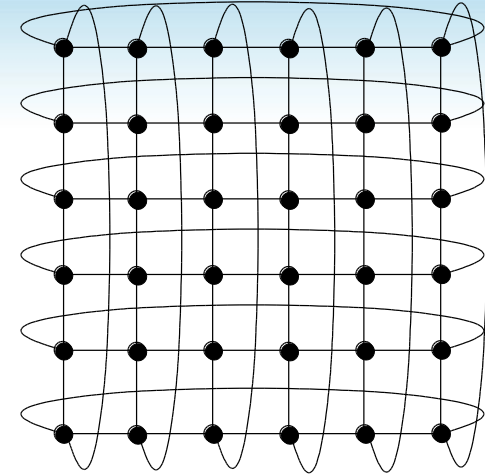
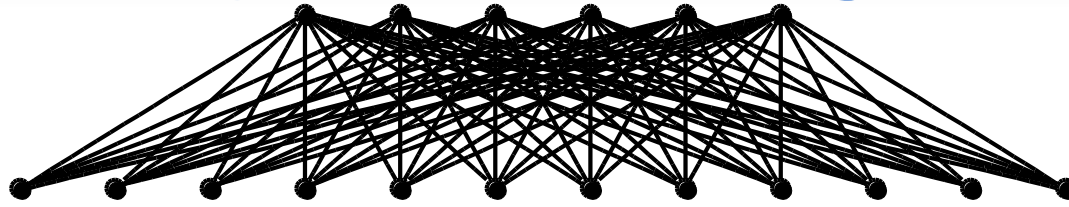


What about Large-Scale Topologies?

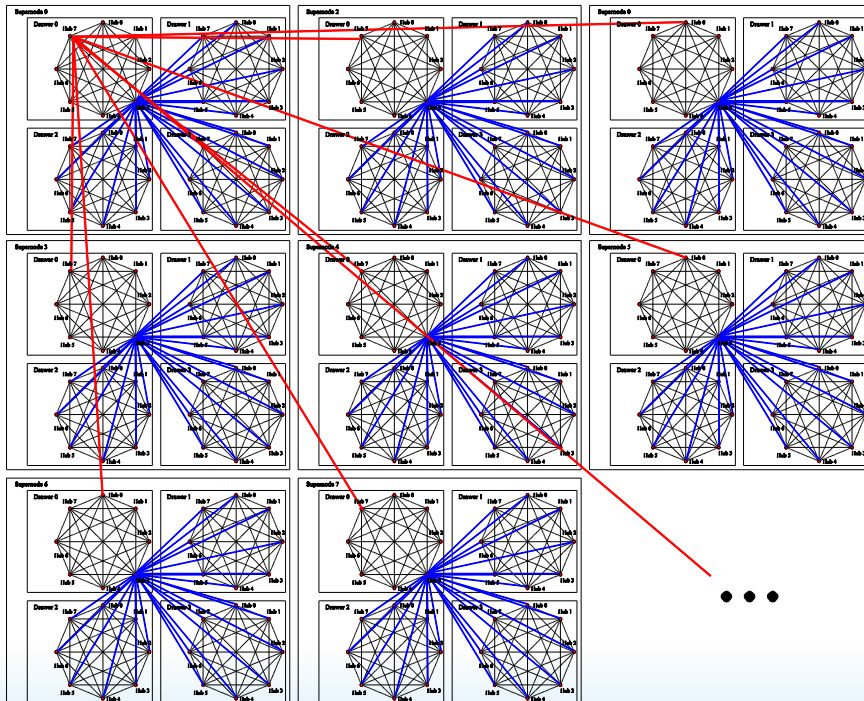
- Fiber optics are most efficient for off-node comm.
 - \approx distance-invariant, number of transceivers count
- **Power consumption**
 - Number of links/lanes
 - Maximum/average hops
- **vs. performance?**
 - Bisection bandwidth (increases number of links)
 - Link bandwidth (increases number of lanes)
 - Node count (increased number of hops)



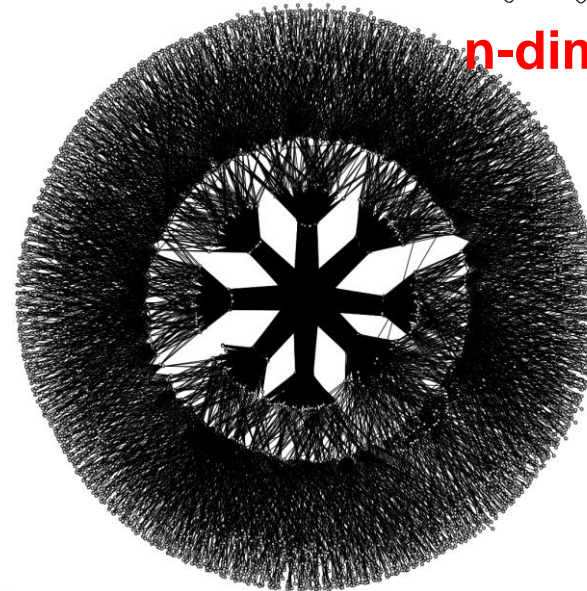
Today's Large-Scale Topologies



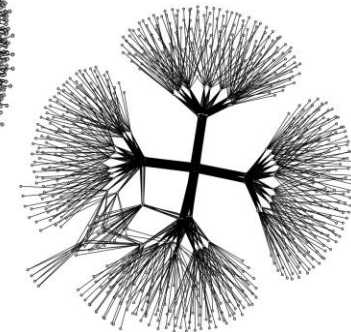
P7-IH/PERCS



Fat-Trees



n-dimensional Tori



Arimilli et al.: The PERCS High-Performance Interconnect

Large-Scale Example Configurations

- 1.3 million PEs, 64 cores each, 80 PEs per node
 - $\sim 2^{14} = 16.384$ network endpoints!

Topology	Number of links	Diameter	Bisection width
Fat-Tree (64 ports, 3 levels)	81.920	6	8.192 (full)
3d-Torus (25x26x26)	50.700	39	1.300 (15.9%)
5d-Torus (8 ⁴ x4)	81.920	18	4.096 (50%)
PERCS	385.024	3	8.192 (full)

Constant cost

(can be reduced with throttling etc.)

Dynamic Cost

(per message costs)

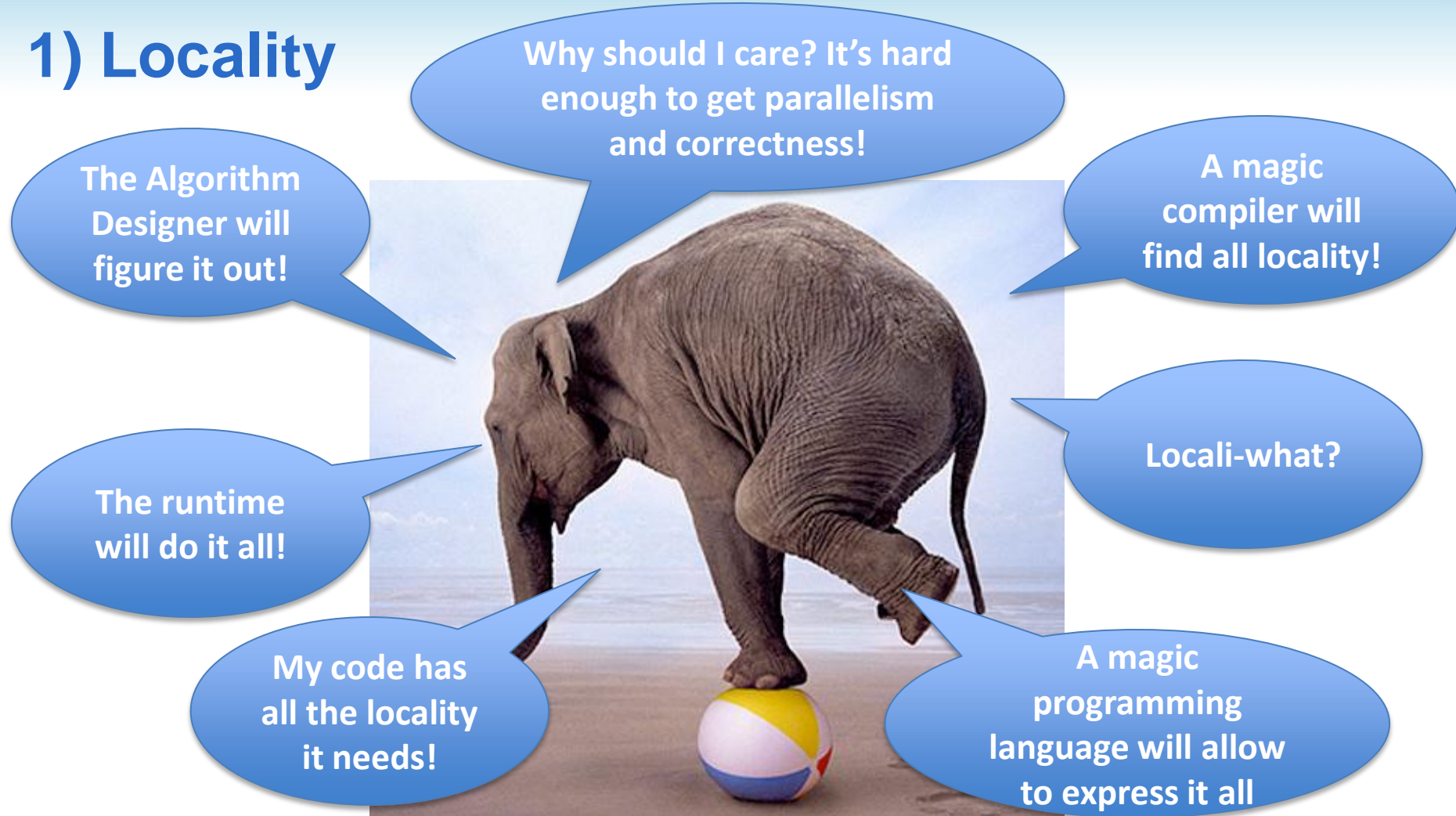


Power-efficient Programming Techniques

1. Locality, locality, locality!
 - Trade-off flops for load/store accesses!
2. Network-Centric Programming
 - Static Optimizations, Overlap
3. Functional specialization
 - Serial accelerators (GPU, FPGA)
 - Network specialization & acceleration
4. Minimize overheads
 - Zero-copy whenever possible!
 - Power-aware middleware



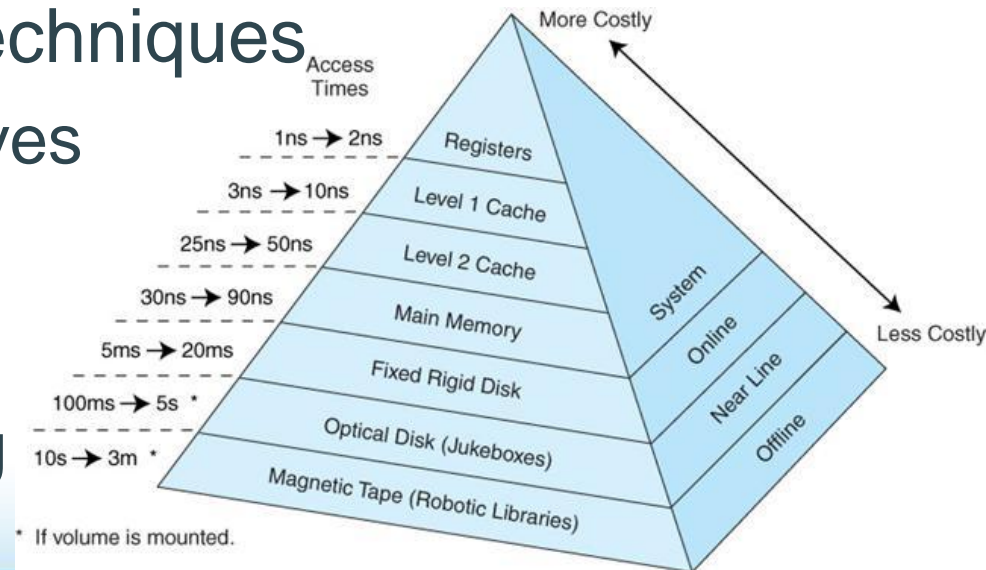
1) Locality



Inspired by A. Snively

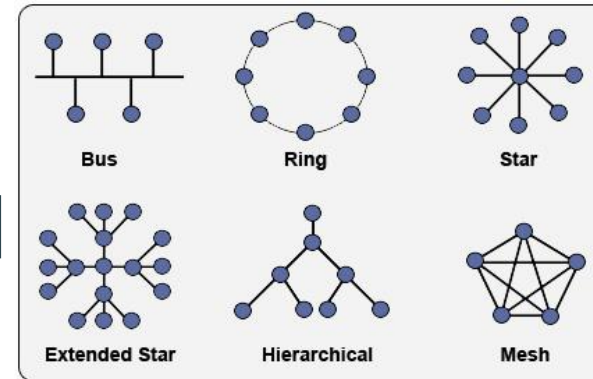
Spatial and Temporal Access Locality

- Cache-aware (or -oblivious) algorithms
 - Well known, sometimes hard to implement
- Well-understood models and metrics
 - Reuse distance
- Well-developed set of techniques
 - Morton ordering, Z curves
- Automation possible
 - Compiler loop-tiling
 - MTL for matrix ordering



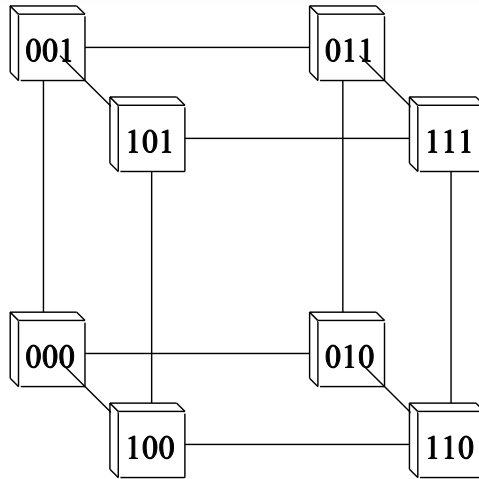
Network Locality

- Mapping relative to network topology, multi-dimensional, hard, NP-complete 😊
 - Very little research, many relevant cases may be polynomial time
- Support in MPI (process topologies)
 - We tackled general case [Hoefler'11]
 - Different optimization goals:
 - Energy consumption (minimize dilation)
 - Runtime (minimize maximum congestion)

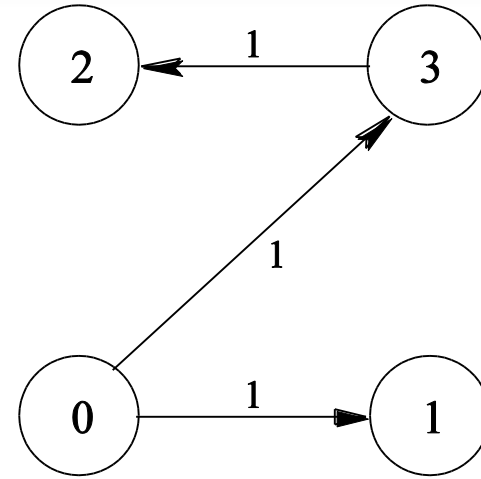


Topology Mapping Example

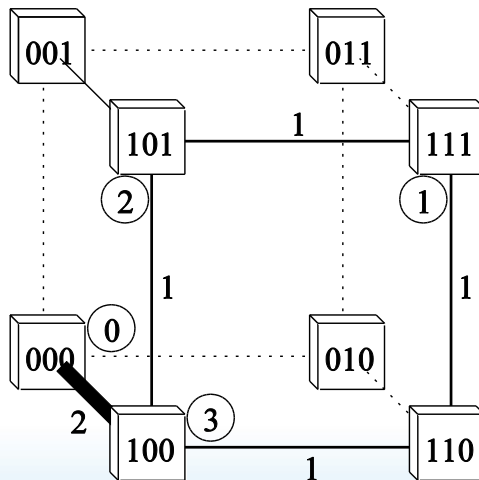
Physical Topology:



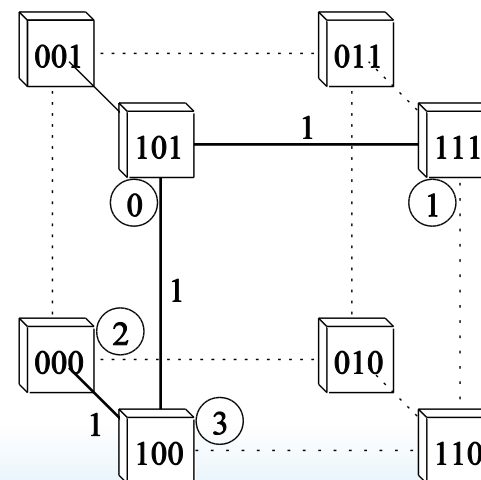
Application Topology:



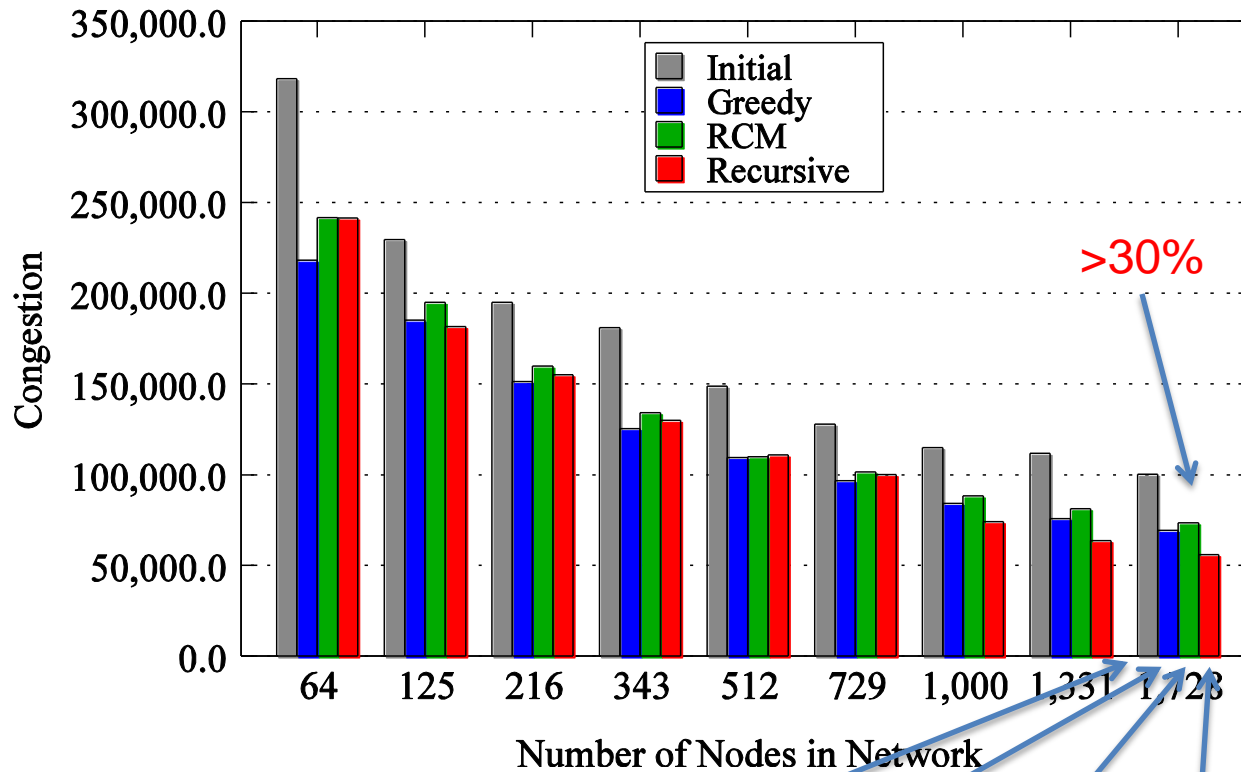
Mapping 1:



Mapping 2:



Topology Mapping Example: 3d Torus



- nlpkkt240, dilation for P=1728: 9.0, 9.03, 7.02, 4.5

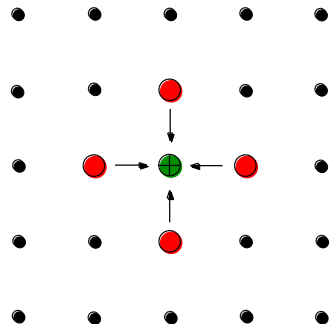
2) Network-Centric Programming

- Make the network **programmable** like a CPU!
 - Application-specific routing
 - Compiler optimizations
 - Static link power management
- What is a good abstraction? Open Research!
 - Need to find a **Network ISA**
 - Our proposal: Group Operation Assembly Language
 - Supports arbitrary communication relations
 - Define GOAL communication graph statically
 - Optimize scheduling and program network

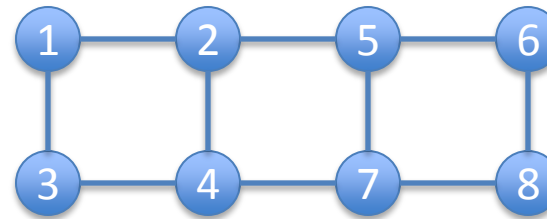


A GOAL Example Program

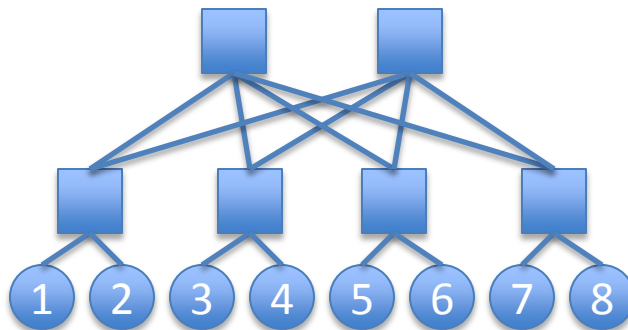
Stencil Computation



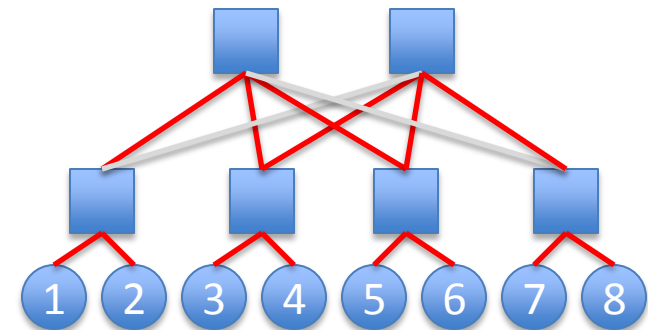
Nearest neighbor communication
Static GOAL Graph:



Fat-Tree Topology



Static Routes and Disabled Links

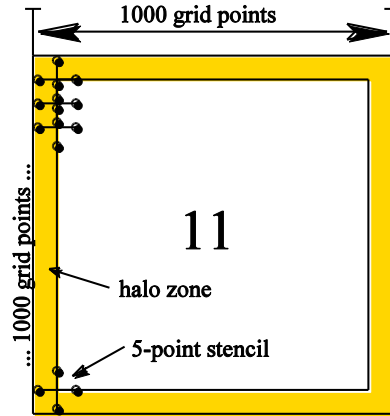
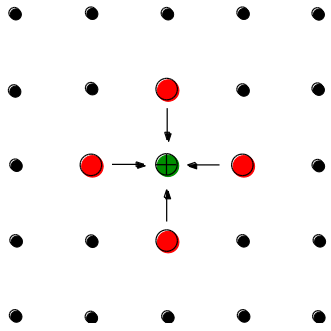


Dualism of Network and CPU Architecture

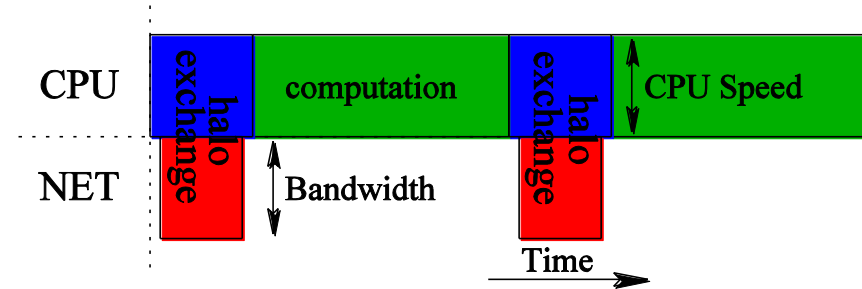
- Similar behavior as CPU architecture
 - Cf. VLSI/EPIC/Vector vs. Multithreaded
- Static programs:
 - Compile routing statically
 - GOAL or sparse collectives in MPI-3.0
- Dynamic programs:
 - Active messages (cf. threads)
 - Cf. Active Pebbles/AM++ [Willcock et al.'11]
- Likely to be a mixture in reality
 - Similar to CPUs with vector and MT instructions!

Keep the Network Busy with Overlap

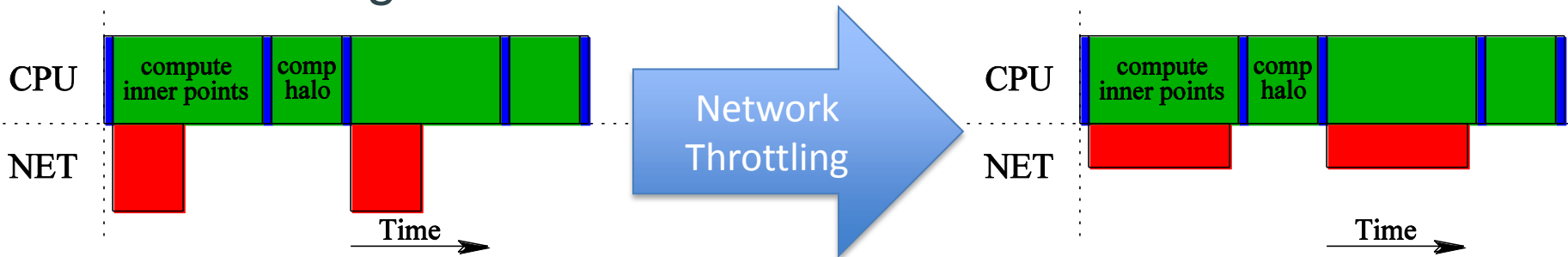
Stencil computation



Blocking Communication



- Nonblocking communication



- Runtime smaller, better energy utilization!

Source: T. Hoefler: Software and Hardware Techniques for Power-Efficient HPC Networking

Minimize Communication Overheads

- Persistent communication
 - Eliminates tag matching
 - Hardware can setup channels
 - MPI_Send_init etc. (needs to be supported!)
- MPI One Sided / PGAS / RDMA
 - Eliminates high-level messaging protocols
 - Direct hardware specialization
- Sparse collectives
 - Specify communication topology statically!

3) Functional Specialization

- We all know about Accelerators
 - I have nothing to add 😊
- Don't forget about FPGAs though
 - Several impressive results for simple problems, e.g., password cracking
- Specialized architectures
 - Anton, MDGrape



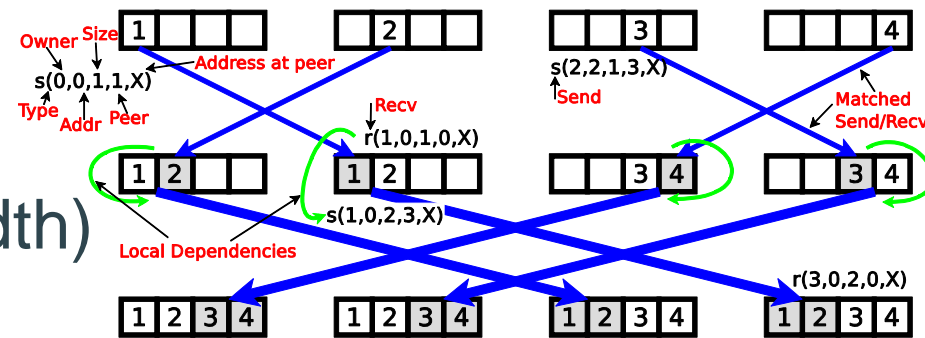
4) Minimize Overheads

- Minimize data movements
 - **Avoid copies**, send/recv from/into user buffers
 - MPI datatypes – [Hoefler'10]
 - Improved performance, reduce energy consumption!
- Energy-optimizing middleware
 - Utilize persistence, program network
 - Energy-aware collective operations
 - Runtime takes the role of the OS [Brightwell'11]

Sources: Hoefler, Gottlieb: Parallel Zero-Copy Algorithms for Fast Fourier Transform and Conjugate Gradient using MPI Datatypes
Brightwell: Why Nobody Should Care About Operating Systems for Exascale

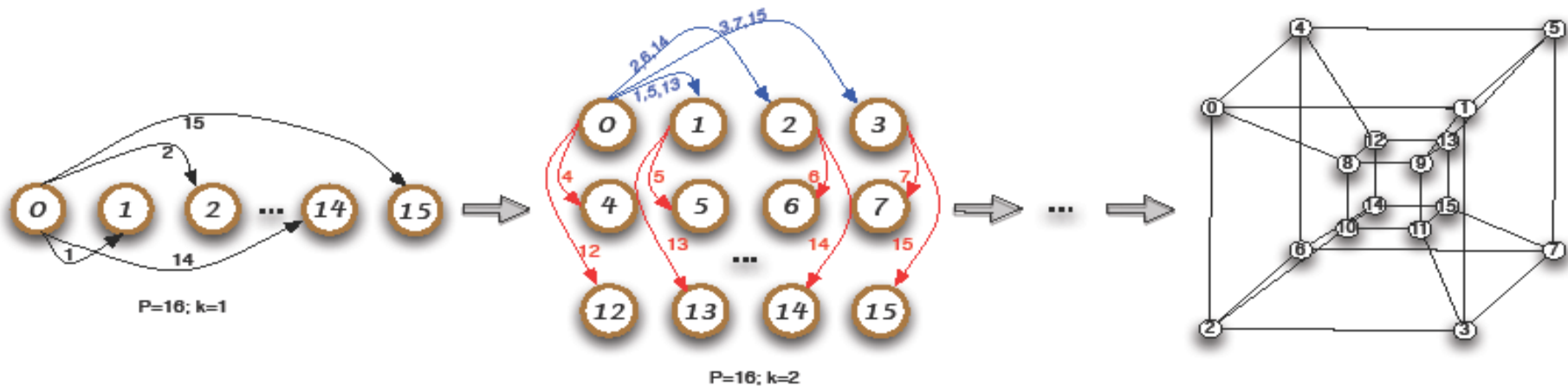
Energy-aware Collective Communication

- Common optimization idiom:
 - Trade excess bandwidth for latency/performance
 - Add additional copies, **increases power**
- Old assumption:
 - Bandwidth is available
 - $T = \max(\text{latency, bandwidth})$
- Performance-optimal:
 - Bruck's algorithm for small data
 - Each item is sent/copied multiple times



Energy-aware Collective Communication

- Assume two components for Energy consumption:
 - The “static” energy: a [J/s]
 - The “dynamic” energy: b [J/B]
 - $E = a * T$ (time) + $b * D$ (transferred data)
- For example Bruck’s alltoall:
 - Choose optimal k for k -d torus ($1 \leq k \leq \log_2 P$)

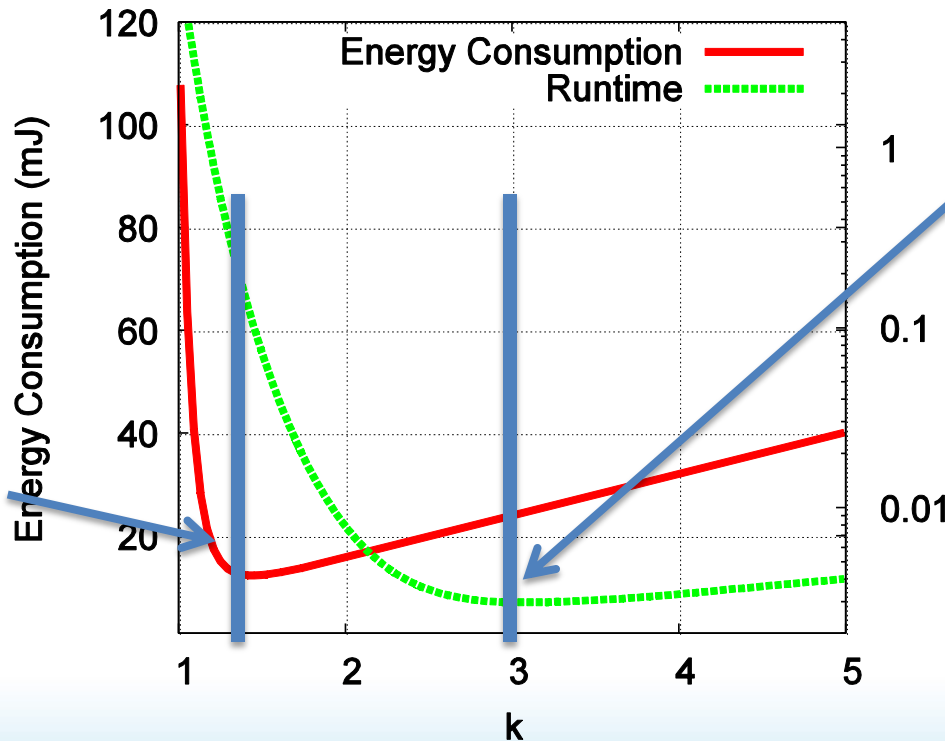


Energy- or Runtime-optimal k for P=10M ?

- $T = k(\alpha \sqrt[k]{P} + \beta S)$ (S = total data size per process)
- $E = aT + bS = ak(\alpha \sqrt[k]{P} + \beta S) + bkS$

$\alpha = 1\mu s$
 $\beta = 0.01ns/B$
 $a = 0.01W$
 $b = 100pJ/B$

Optimal Energy
 16mJ
 (Runtime=0.8s)



Optimal Runtime
 0.003s
 (Energy=25mJ)

$P = 10M$
 $S = 80MB$

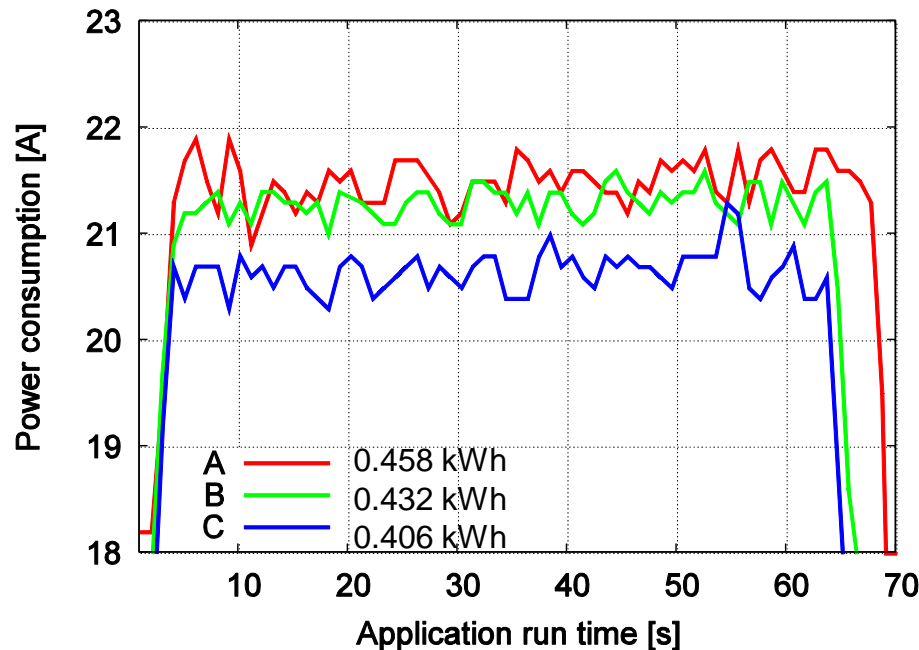
Summary: Energy-aware Programming

- Optimize for power-consumption, not speed
 - Often close but not always! Stop counting Flops!
- Needs a good **model** of power consumption for algorithm designers (data movement?)
- Needs **measurement tools**/hooks for software designers (“energy counters”)
- Power analysis and monitoring tools
- → extend performance tools with power metrics!
 - Important ongoing work!

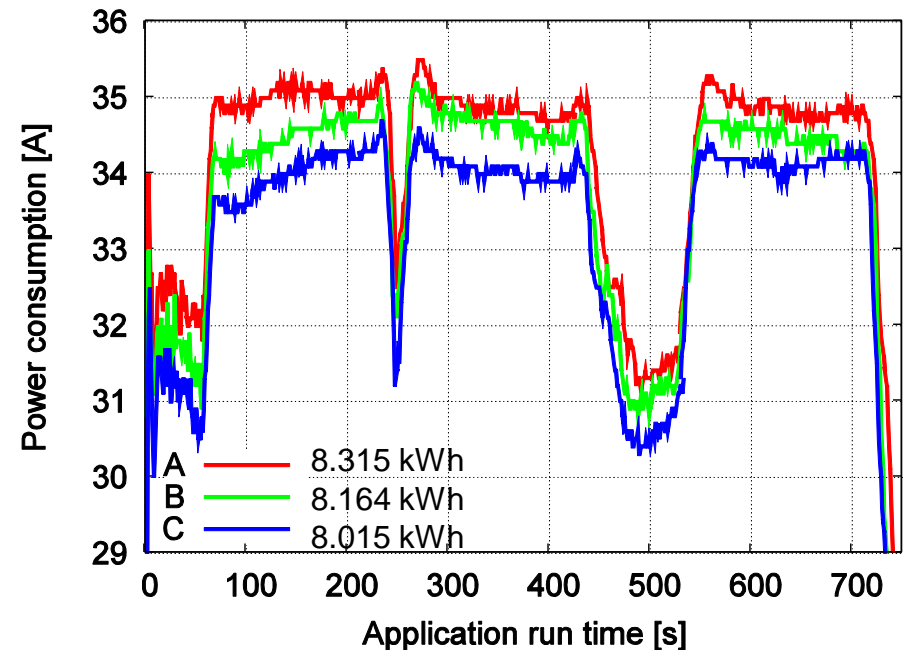


We need more Data! Especially on Networks!

- Studied application power consumption with different networks (A- IB/C, B – MX/C, C – MX/F)



Parallel Ocean Program



RAXML

Source: Hoefler, Schneider et al.: A Power-Aware, Application-Based, Performance Study Of Modern Commodity Cluster Interconnection Networks

A Quick Glance at Analytic Power Modeling

- Similar to performance modeling, observe power instead of time though!
 - Analytic ab-initio modeling is hard (needs very detailed power models)
 - Empirical modeling seems feasible (needs measurement support for power consumption)
- Analyze tradeoffs between architectures
 - Simple vs. complex cores, co-design, detailed feasibility studies with key applications, complex minimization problem

Thanks and Summarizing!

- Main routes to follow in the near future:
 - Improve locality/reduce communication (at all levels!)
 - Regulate power consumptions of subcomponents
 - Explicit design (scratchpad, network-centric progr.)
 - Overlap and balance (parallelism \uparrow)
- Techniques/Research Directions:
 - Network topologies (low distance)
 - Power-aware algorithms (I/O cplx)
 - Power analysis and modeling

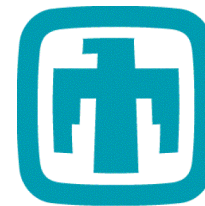


Collaborators, Acknowledgments & Support

- Thanks to:
 - Marc Snir, William Gropp, Wen-mei Hwu
 - Vladimir Voevodin, Anton Korzh for comments
- Sponsored by



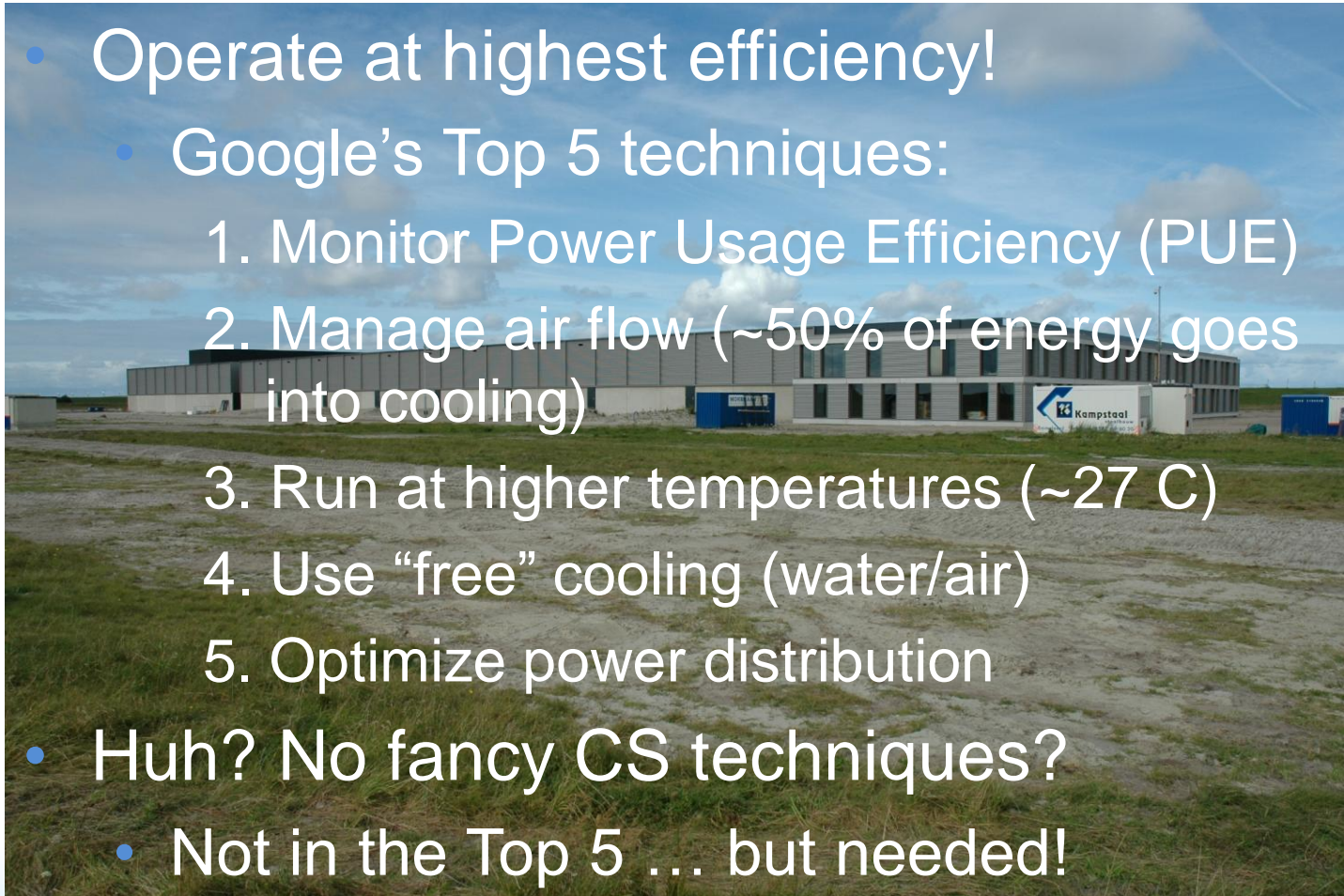
ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN



**Sandia
National
Laboratories**

Google, the datacenter energy pioneers?

- Operate at highest efficiency!
 - Google's Top 5 techniques:
 1. Monitor Power Usage Efficiency (PUE)
 2. Manage air flow (~50% of energy goes into cooling)
 3. Run at higher temperatures (~27 C)
 4. Use "free" cooling (water/air)
 5. Optimize power distribution
- Huh? No fancy CS techniques?
 - Not in the Top 5 ... but needed!

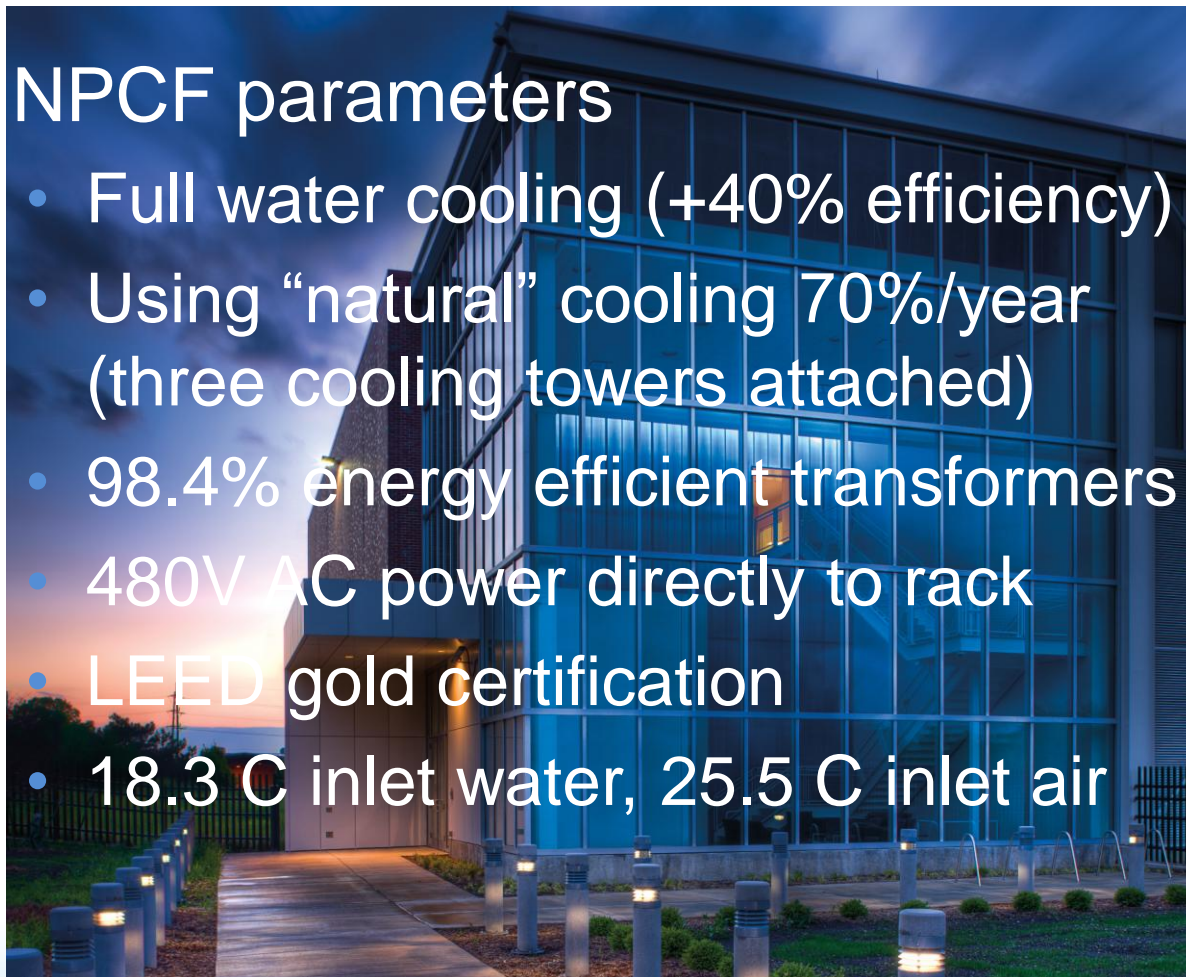


Source: Google

HPC Centers Operate Large Datacenters too

NPCF parameters

- Full water cooling (+40% efficiency)
- Using “natural” cooling 70%/year (three cooling towers attached)
- 98.4% energy efficient transformers
- 480V AC power directly to rack
- LEED gold certification
- 18.3 C inlet water, 25.5 C inlet air



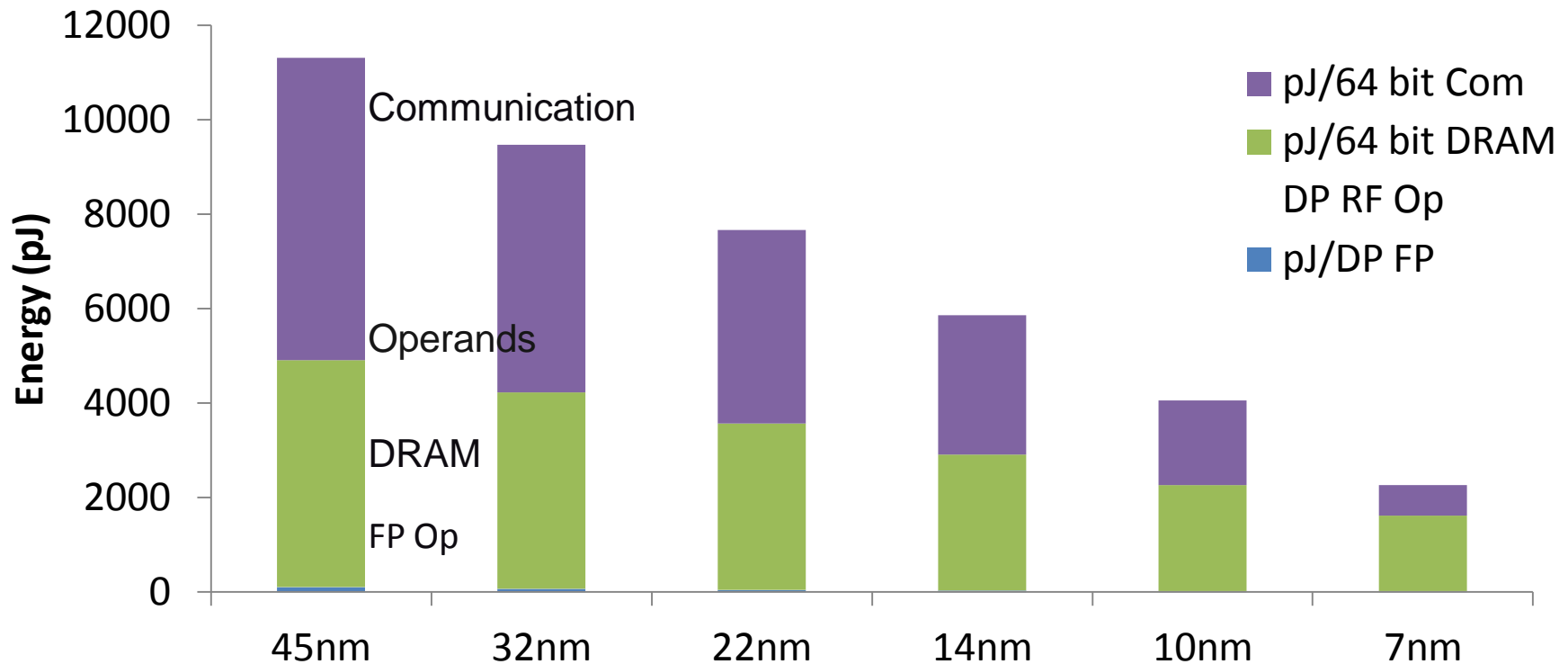
Today's Power Breakdown (w/o overheads)

Operation (64 bit ops)	Energy (pJ)	FP ADD: a=b+c	DP FLOP ratio
FP FMA (2 FLOPs)	100	50	1
INT Add	1	-	-
Register (64x32 bank)	3.5	10.5	0.2
SRAM (64x2k)	25	75	0.67
Move 1mm	6	18	2.78
Move 20mm	120	360	7.2
Move off-chip	256	768	15.36
DRAM	2000	6000	120

Source: Dally, 2011

- Operation cost will shrink with feature size
- DRAM cost will shrink with architectural changes
- **Movement costs are hard to reduce!**

Predictions for Scaling the Silicon



- Assuming no architectural changes (DRAM will likely be even lower)

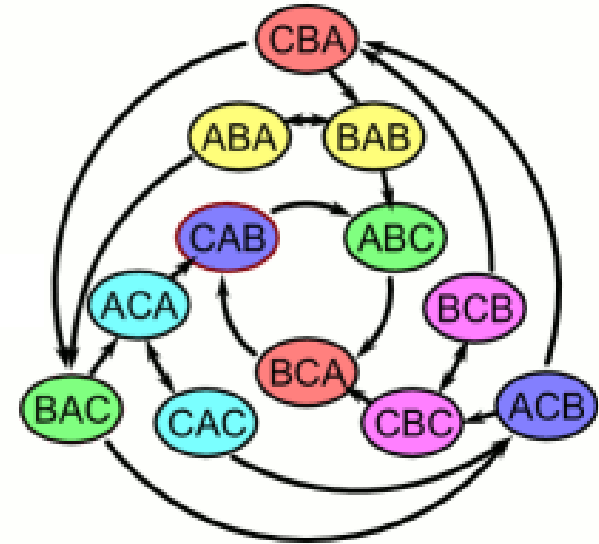
Source: S. Borkar, Hot Interconnects 2011

All those are lower bounds!

- Ideal cache, ideal CPU ...
 - Need to avoid any additional overheads
 - Need simpler CPU architectures
 - Caches have a huge energy-saving potential!
- The network may be much more important!?
 - Not discussed so far at all!
 - I/O complexity works well with networks too
 - **Local memory modeled as “cache”**

The Quest for Low-Diameter Networks

- Low diameter \rightarrow low power
- High-radix routers \rightarrow high power and cost
- Fundamental limit for radix- r routers and n nodes
 - diameter $\geq \approx \log_r(nr)$
- Minimize energy by trading off:
 - Router radix (r) with diameter
- Faces *degree-diameter problem* for optimal solution



Power-aware Programming

- Now we have (lower-bound) **hardware** and **algorithmic** solutions
 - We can still loose infinite power in the implementation 😊
- Power-aware programming is most important!
 - Simple observation: using the machine more efficiently **decreases power consumption** and **increases performance!** (non-conflicting optimization goals!)
 - Why? Idle resources consume power too (~10%)

2) Network-centric Programming

- Overlap, overlap, overlap
 - Keep memory, CPU, and network busy
 - More parallelism needed ☹️
- Prefetch memory
 - Hardware prefetcher in modern architectures
 - May waste power!
 - Explicit prefetching! Compiled in or as SMT thread



MPI Topology Mapping

- Application topologies are often only known during runtime
 - Prohibits mapping before allocation
 - Batch-systems also have other constraints!
- MPI-2.2 defines interface for re-mapping
 - Scalable process topology graph
 - Permutes ranks in communicator
 - Returns “better” permutation π to the user
 - User can re-distribute data and use π



A Reward for the Careful Analysis



- Cluster Challenge 2008 winners: Dresden/Indiana

Why do we HPC folks care about energy?

- Our requirements are on exponential scaling too
 - “Expect” to double “performance” every 18 months at roughly equal costs (including power)
 - As we all know, this is more complex and we’re facing the “Multicore Crisis” or in HPC “Scalability Crisis”
 - Managing billion-way parallelism (?)
- Not only frequency scaling stopped!
 - Voltage scaling stopped
 - Traditional architectural advances kill power budget
 - Large-scale computing will hit the “Energy Crisis” soon

Network Acceleration

- Message handling in hardware
 - Pipelining (done by most networks)
 - Message Matching (CAMs vs. list traversal)
- Collective operation offload
 - saves bus transactions (improves “locality”)
 - specialized execution, avoid copies
 - Examples: GOAL, Portals, ConnectX2
- Programmable networks
 - To be developed!



Energy- or Time-Optimal Blocking?

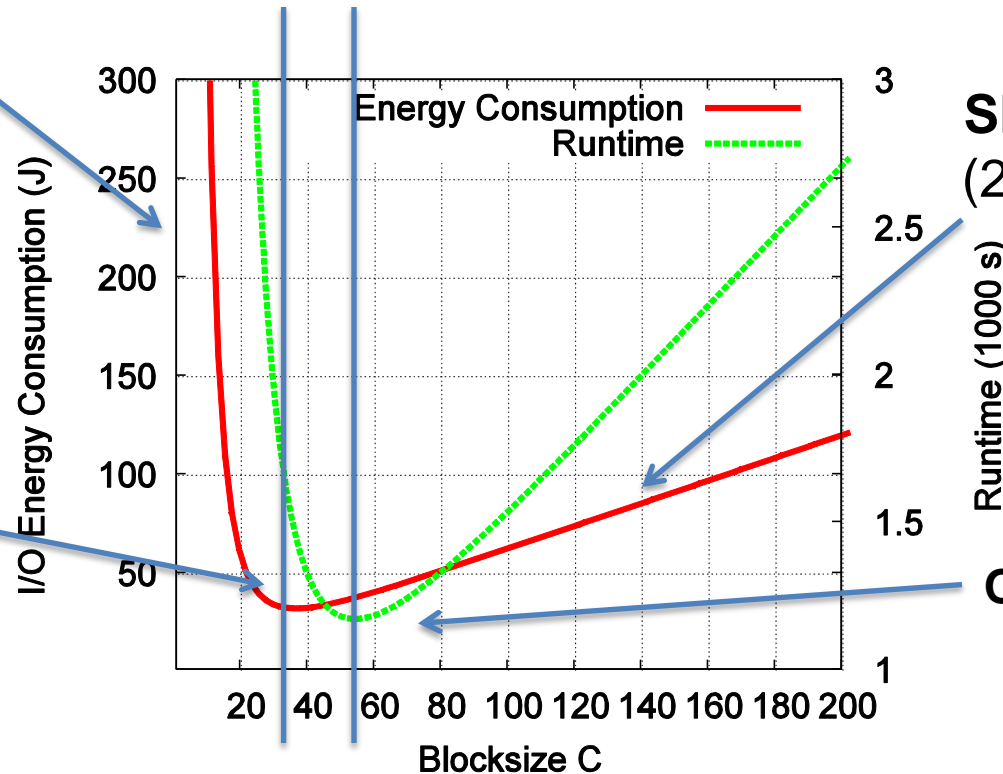
- Assuming single-level hierarchy (ignoring register)

DRAM dominated
 $(2N^2/C^3 + N^2) * 1 \text{ nJ}$

SRAM dominated
 $(2N^2C + N^2) * 0.1 \text{ nJ}$

Optimal Energy

Optimal Runtime



- Non-obvious optimization, derive & repeat