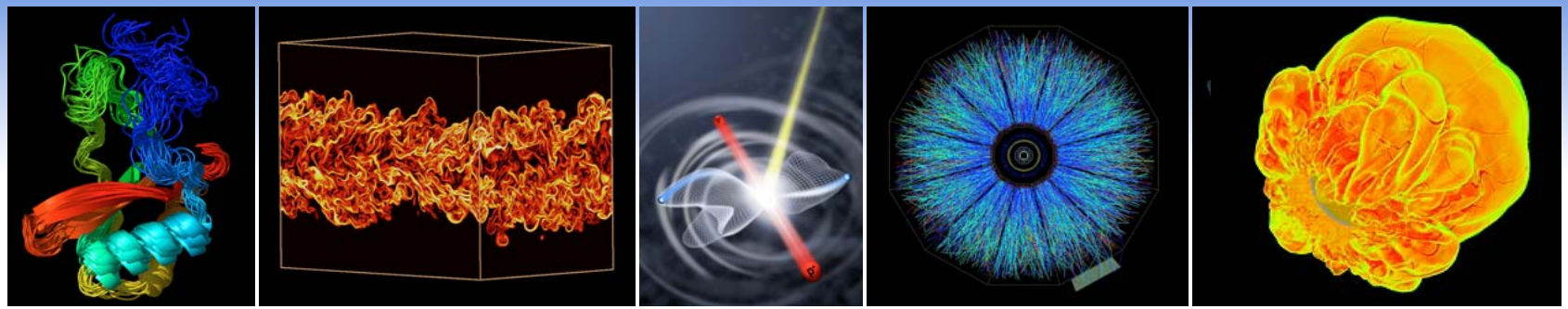
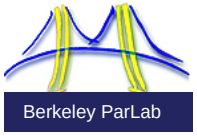


Ten Ways to Waste a Parallel Computer

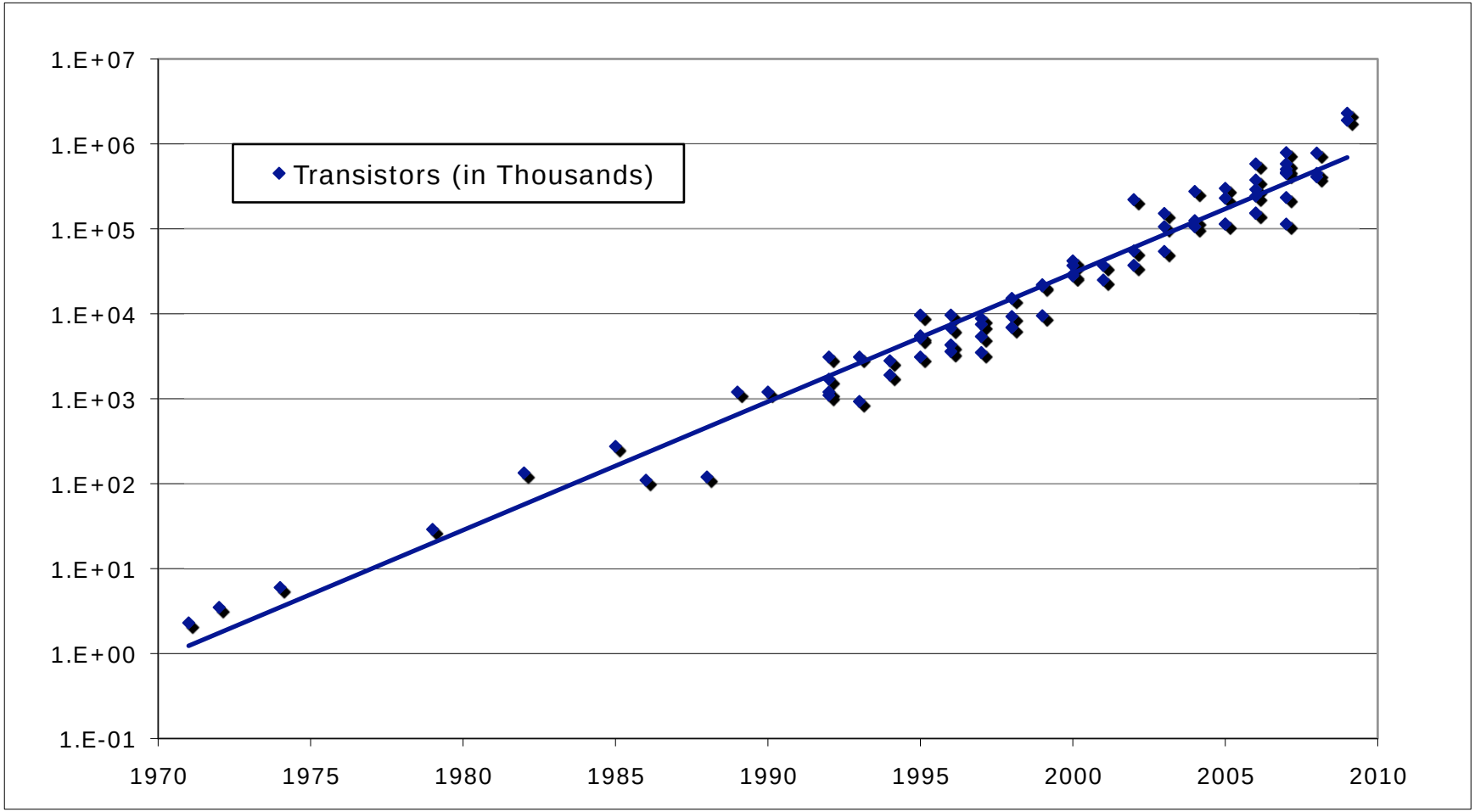
Kathy Yelick

NERSC Director, Lawrence Berkeley National Laboratory

EECS Department, UC Berkleey



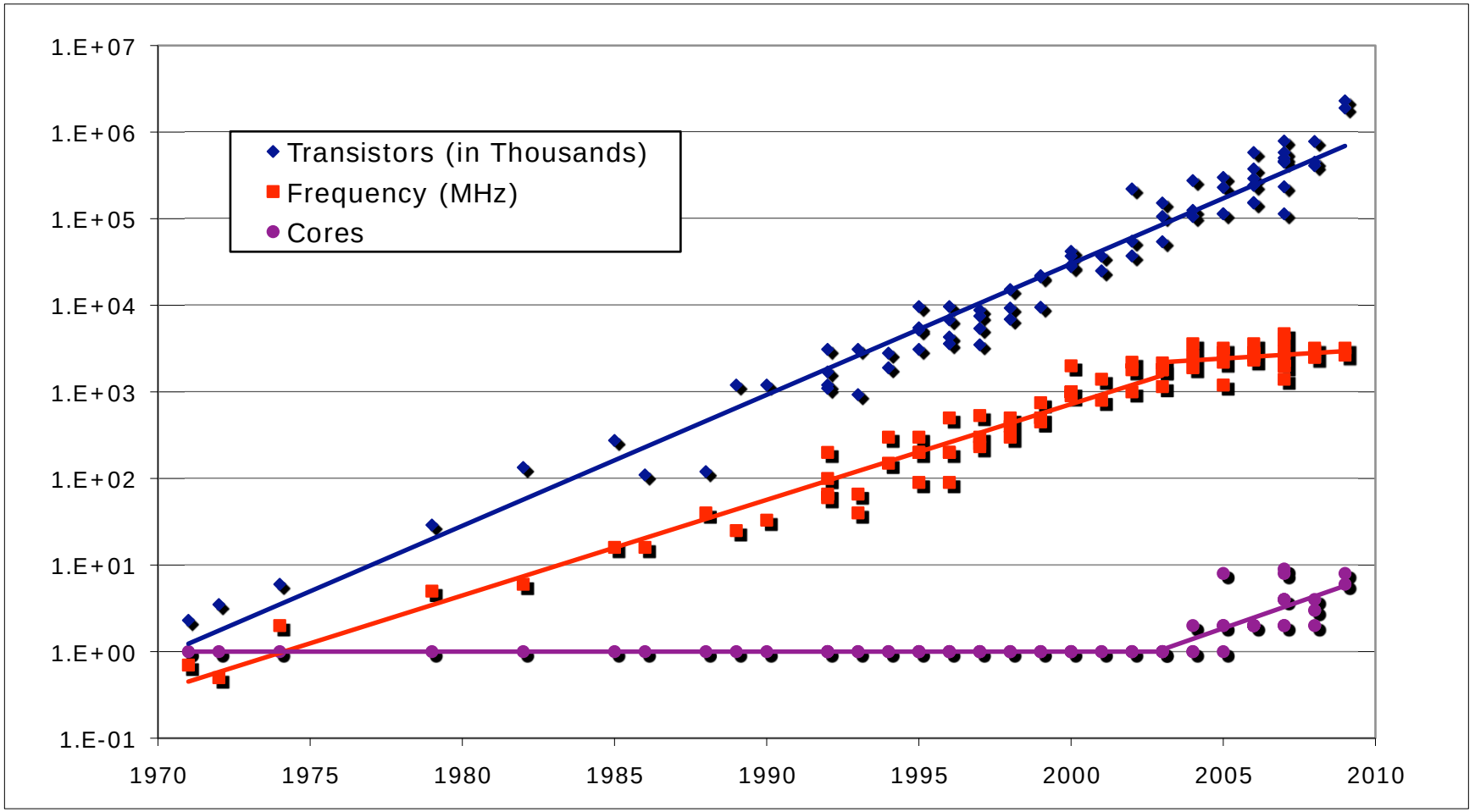
Moore's Law is Alive and Well



Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanoviç

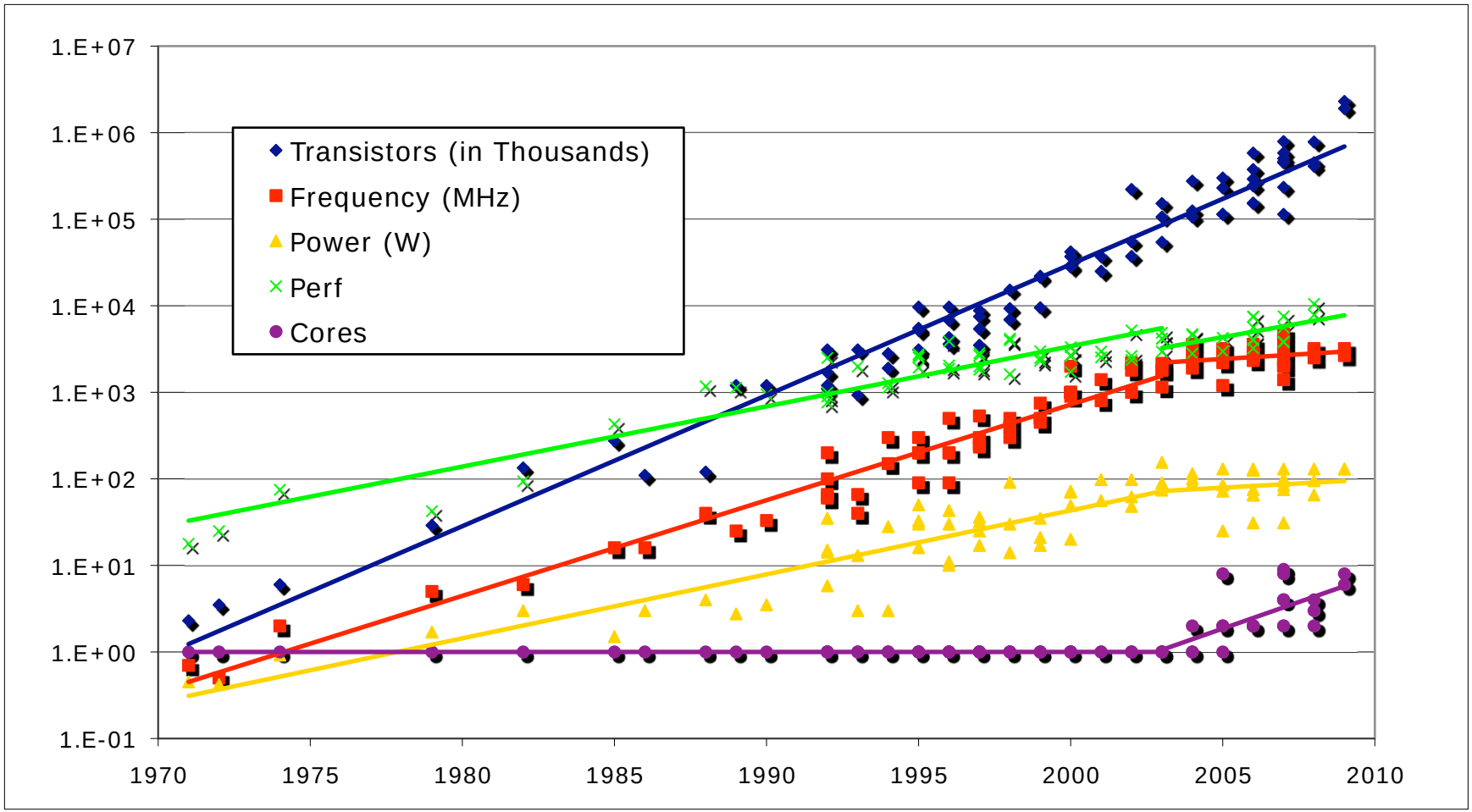
Berkeley ParLab

But Clock Frequency Scaling Has Been Replaced by Scaling Cores / Chip



Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanoviç

Performance Has Also Slowed, Along with Power (the Root Cause of All This)

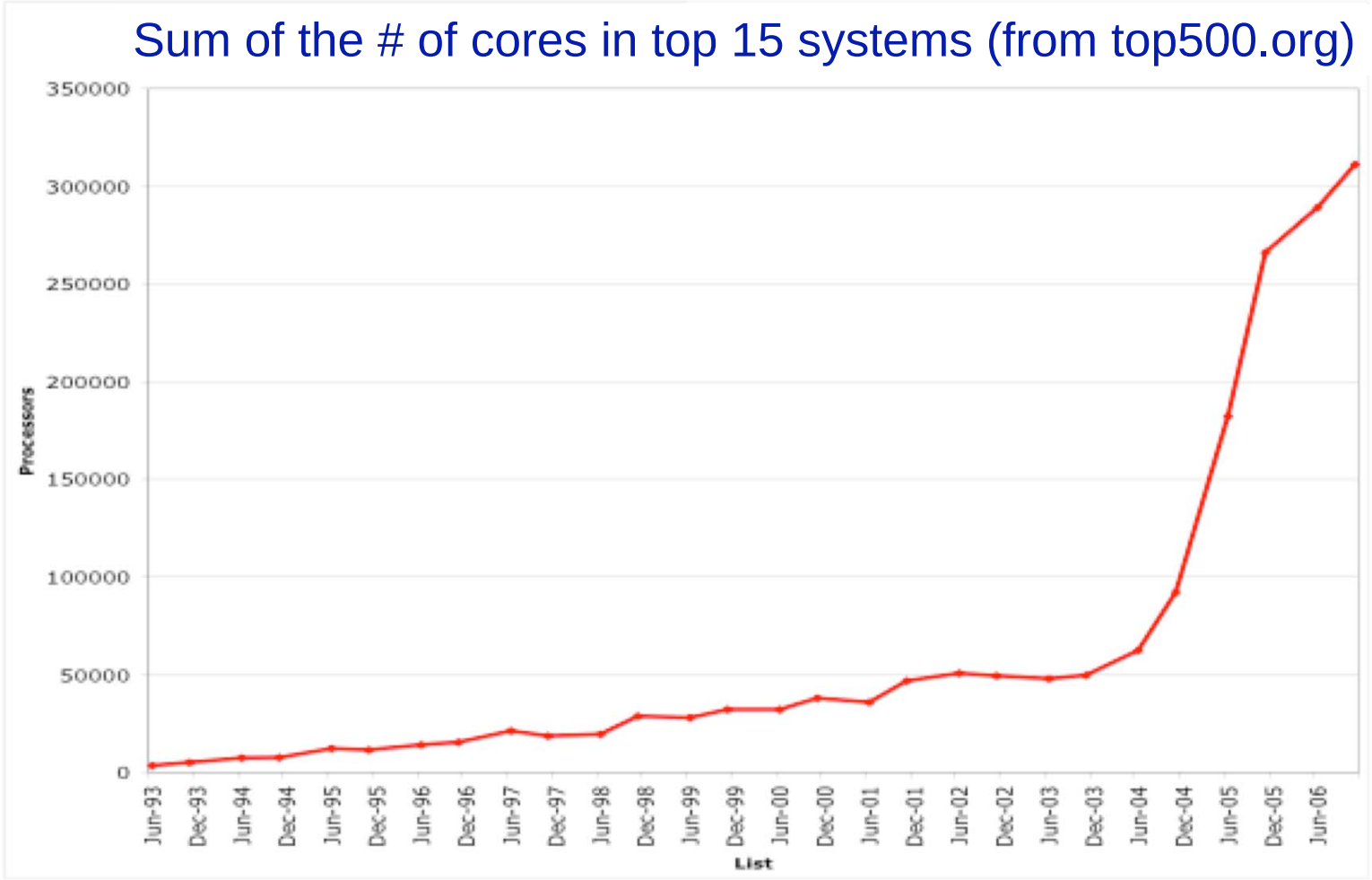


Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanoviç

Berkeley ParLab

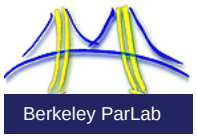
This has Also Impacted HPC System Concurrency

Sum of the # of cores in top 15 systems (from top500.org)



Exponential wave of increasing concurrency for forseeable future!

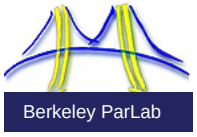
1M cores sooner than you think!



New World Order

- **Goal: performance through parallelism**
- **Power is overriding hardware concern:**
  - **Power density limits clock speed**
  - **Handheld devices limited by battery life**
  - **HPC systems may be >100 MW in 10 years**
- **Performance is now a software concern**

- ***How can we lose performance and therefore lose the case for parallelism?***



#1: Build Systems with Insufficient Memory Bandwidth

- **Memory Bandwidth Starvation**

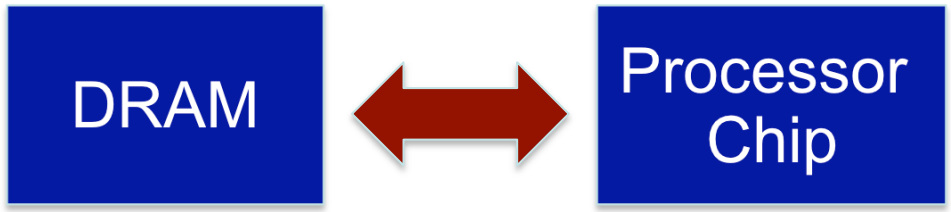
  "Multicore puts us on the wrong side of the memory wall. Will CMP ultimately be asphyxiated by the memory wall?"

  --Thomas Sterling

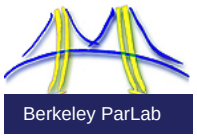
- **Simple double buffering model uses only**
  - **Time to fill up all on-chip memory**

    memory size / bandwidth
  - **Time to compute on all on-chip data**

    memory size bytes * *algorithmic intensity* / ops-per-sec

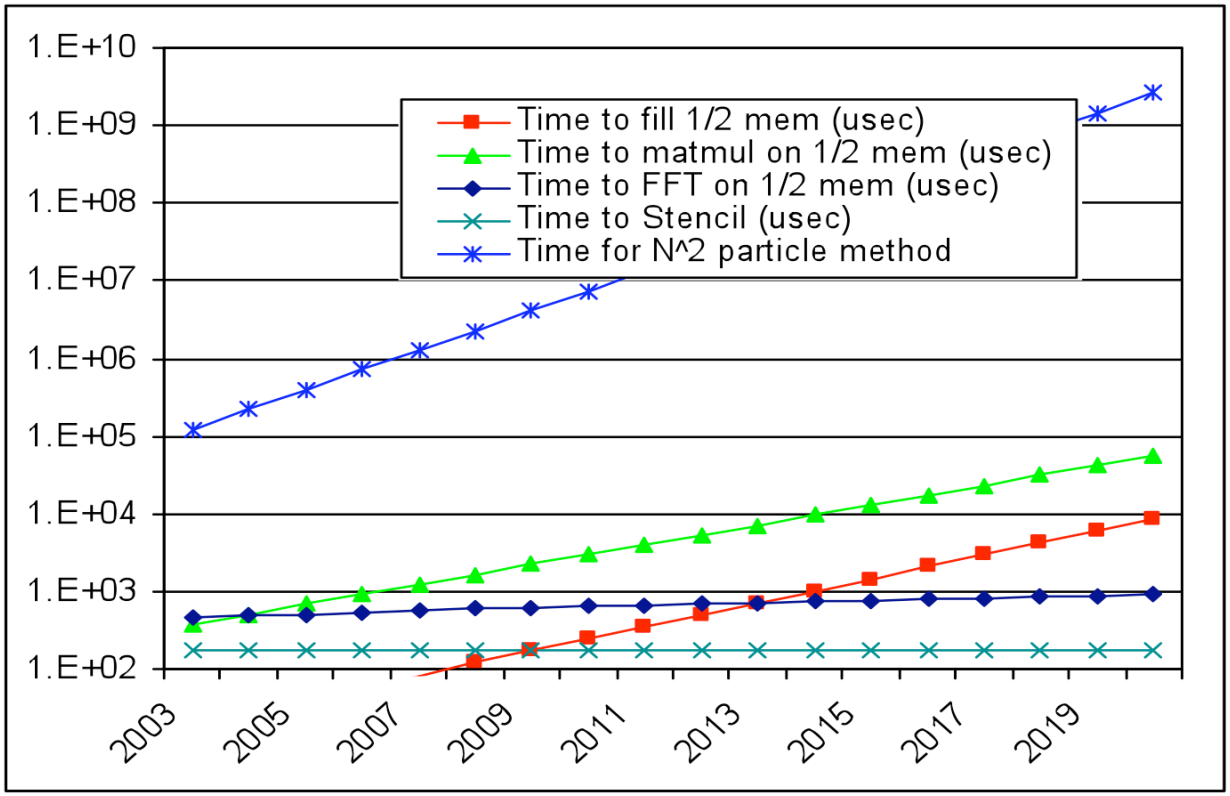


Nothing new, except that ops-per-sec on a chip (aggregate) is still going up

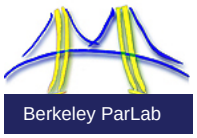


#1: Build Systems with Insufficient Memory Bandwidth

- Under some assumptions about scaling over time
- Can determine for a given algorithm class (constants matter!) when you are bandwidth-limited



Technology to solve this problem if there is market pressure

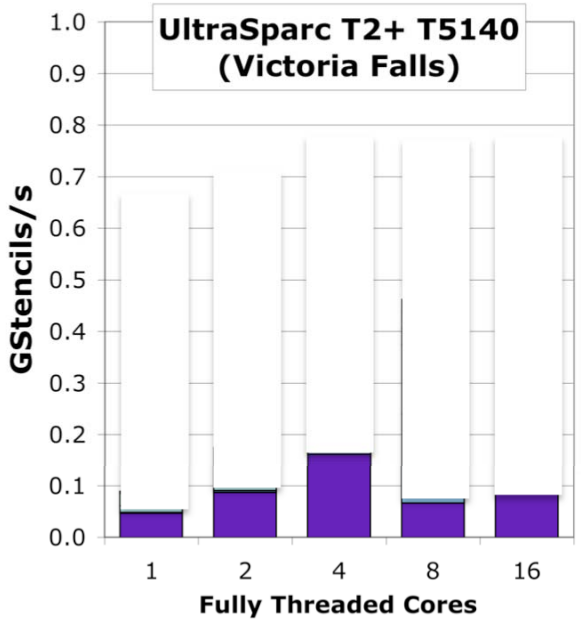


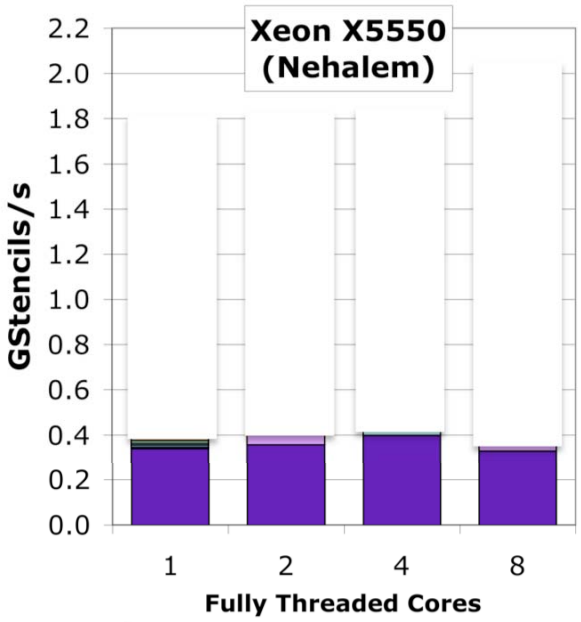
#2: Don't Take Advantage of Hardware Performance Features

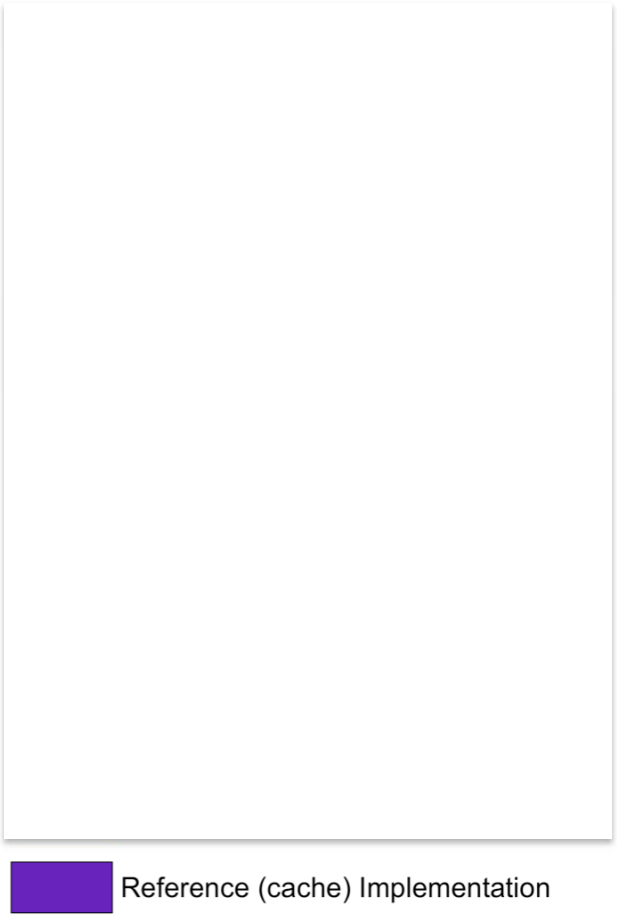
Nearest-neighbor 7point stencil on a 3D array

Use Autotuning!

Write code generators and let computers do tuning



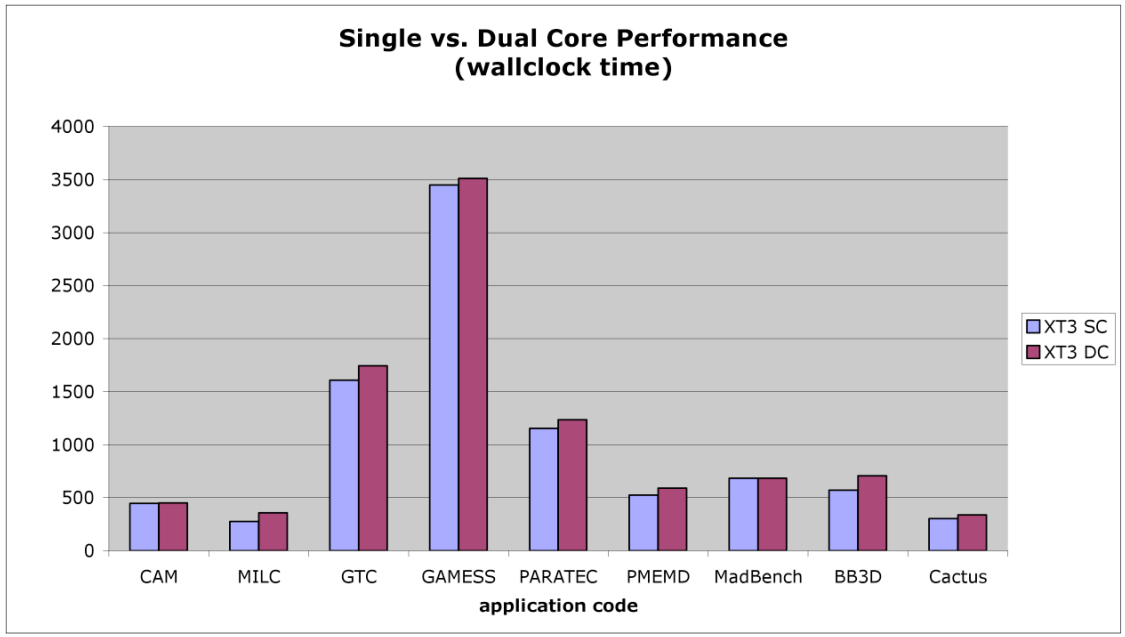




#3: Ignore Little's Law

Little's Law: required concurrency = bandwidth * latency

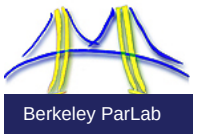
#outstanding_memory_fetches = bandwidth* latency



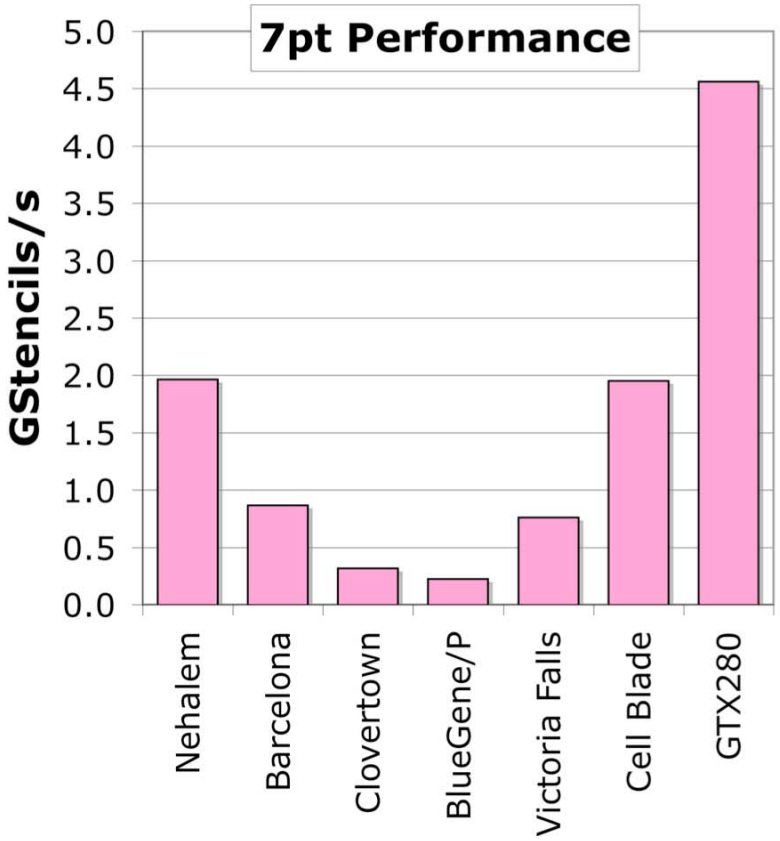
NERSC application benchmarks

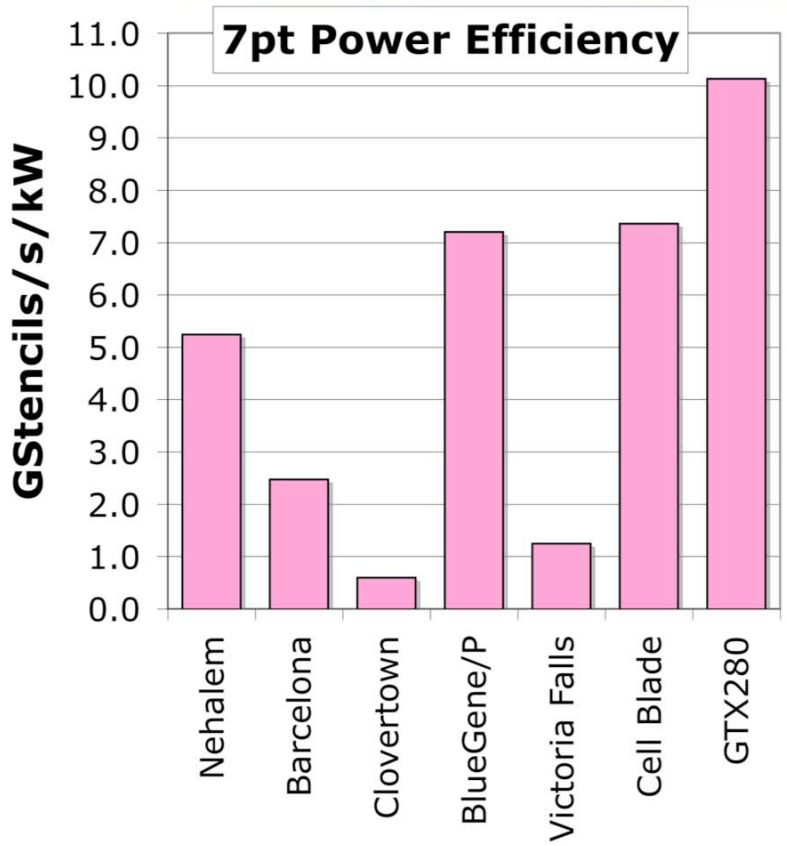
Shalf et al

- **Experiment: Running on a fixed number of cores**
  - **1 core per socket vs 2 cores per socket**
- **Only 10% performance drop from sharing (halving) bandwidth**

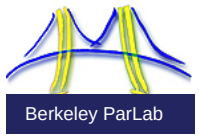


7 Point Stencil Revisited



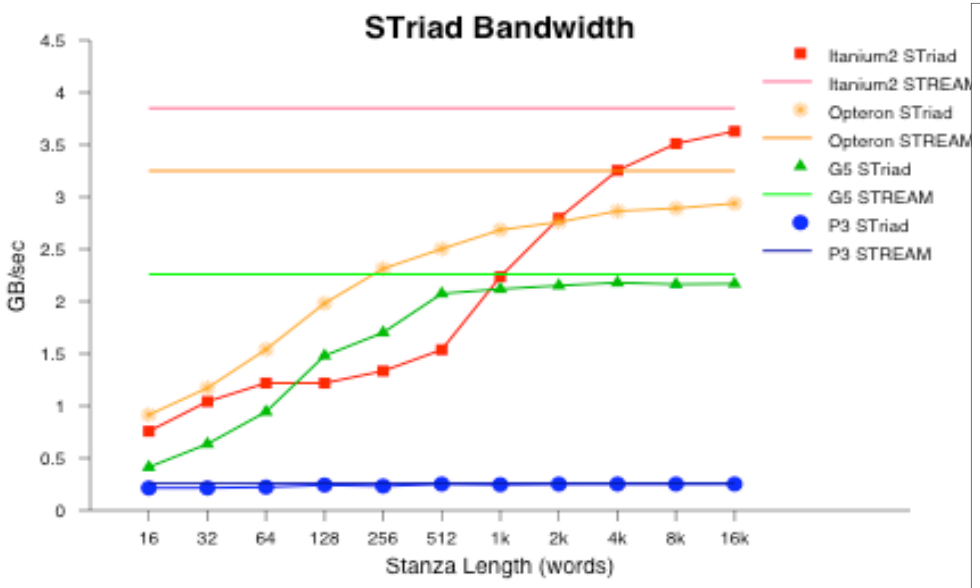


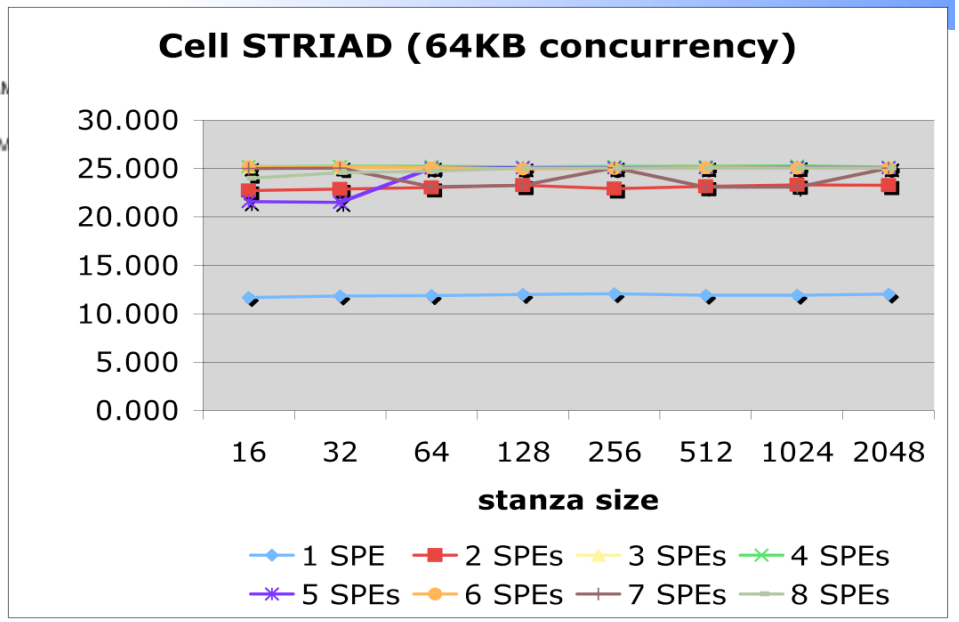
- Cell and GTX280 are notable for both performance and energy efficiency



Joint work with Kaushik Datta, Jonathan Carter, Shoaib Kamil, Lenny Oliker, John Shalf, and Sam Williams

Why is the STI Cell So Efficient?





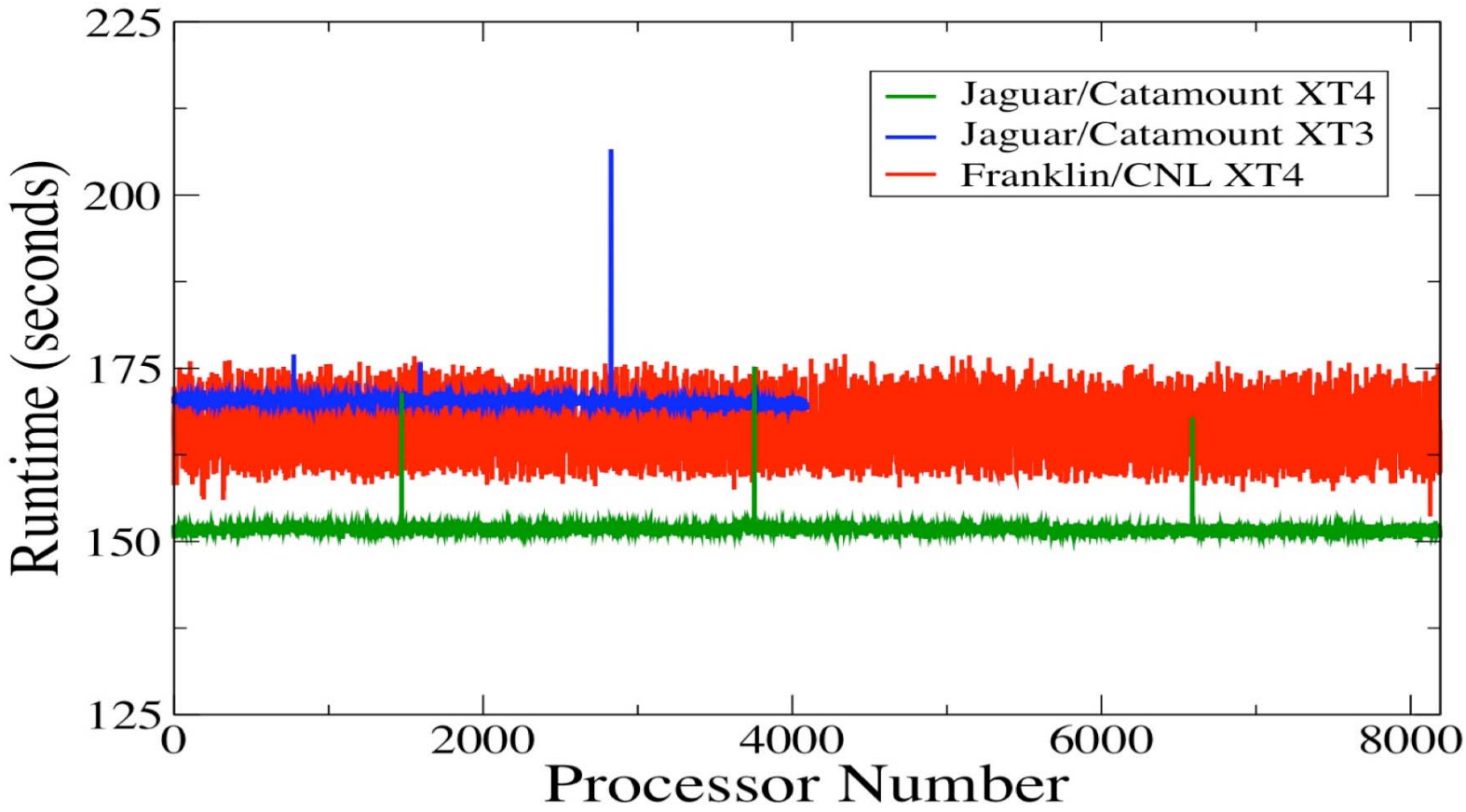
- Unit stride access is as important as cache utilization on processors that rely on hardware prefetch
  - Tiling in unit stride direction is counter-productive: improves reuse, but kills prefetch effectiveness
- Software controlled memory gives programmers more control
  - Spend bandwidth on what you use; bulk moves (DMA) hide latency

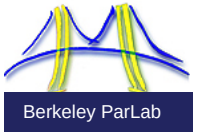
Berkeley ParLab

Joint work with Shoaib Kamil, Lenny Oliker, John Shalf, Kaushik Datta

#4: Turn Functional Problems into Performance Problems

- **Fault resilience introduces inhomogeneity in execution rates *(error correction is not instantaneous)***



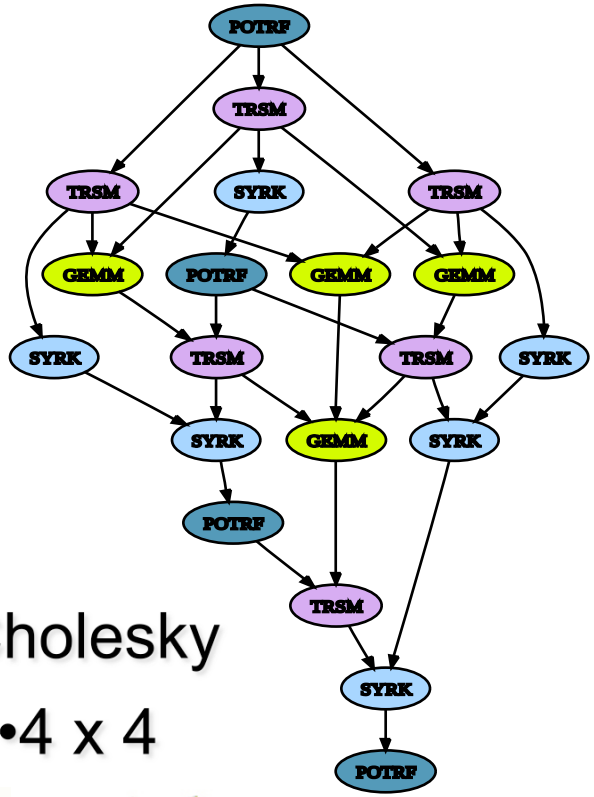


Slide source: John Shalf

#5: Over Synchronize Applications

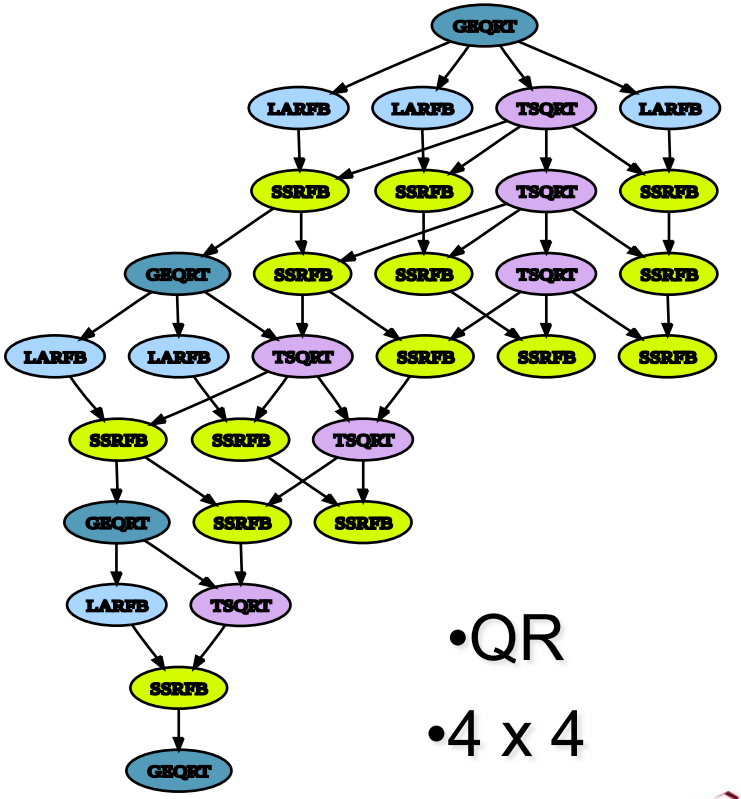
Computations as DAGs

View parallel executions as the directed acyclic graph of the computation



•Cholesky

•4 x 4



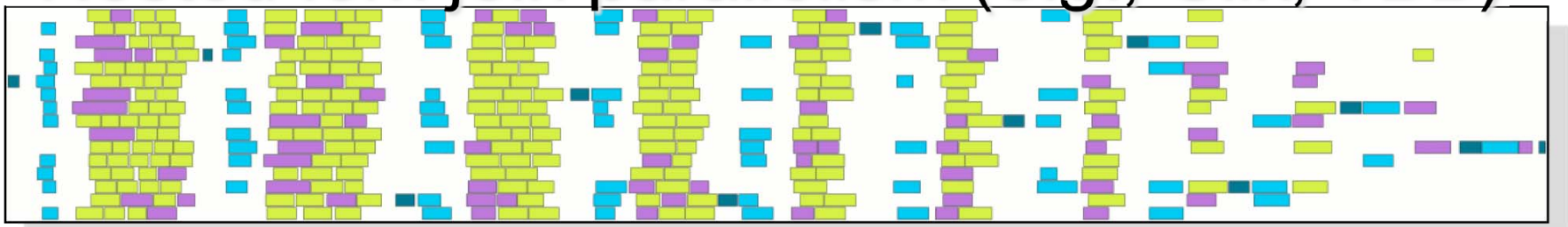
•QR

•4 x 4

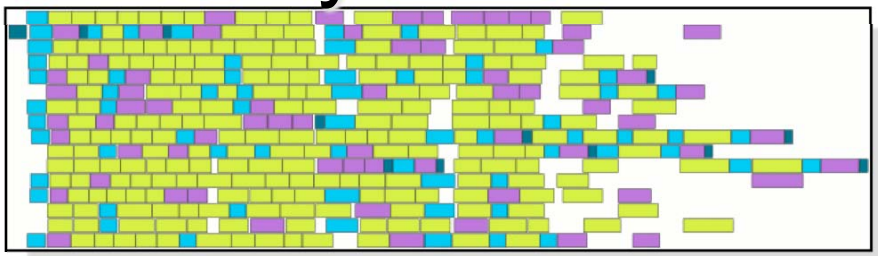
Slide source: Jack Dongarra

Cholesky using PLASMA

- Nested fork-join parallelism (e.g., Cilk, TBB)

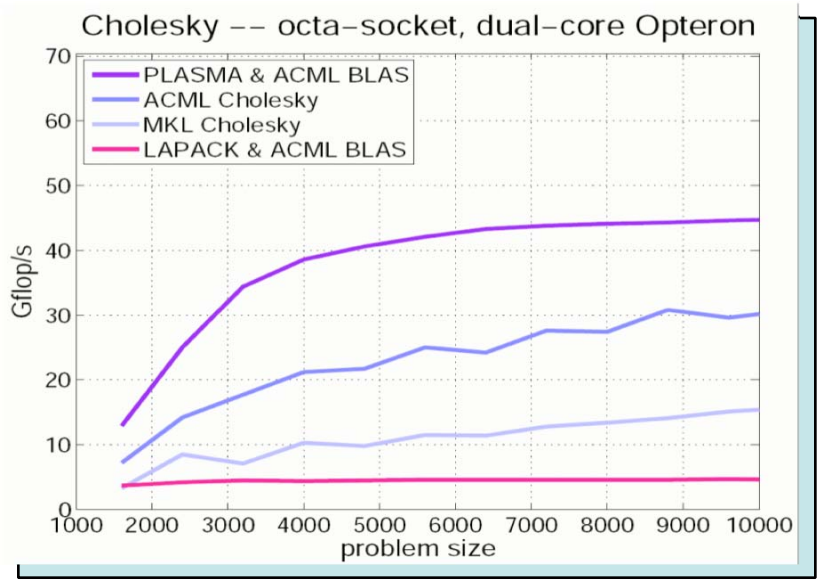


- Arbitrary DAG scheduling (e.g., PLASMA, SuperMatrix)

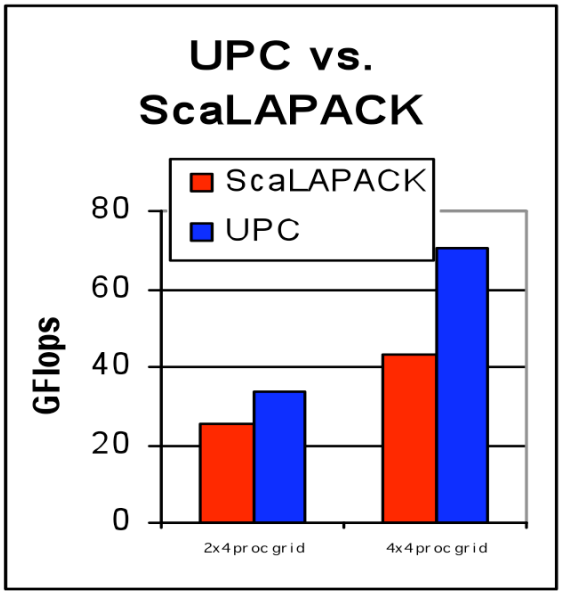


DAG Scheduling Outperforms Bulk-Synchronous Style

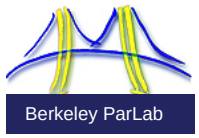
PLASMA on shared memory



UPC on partitioned memory



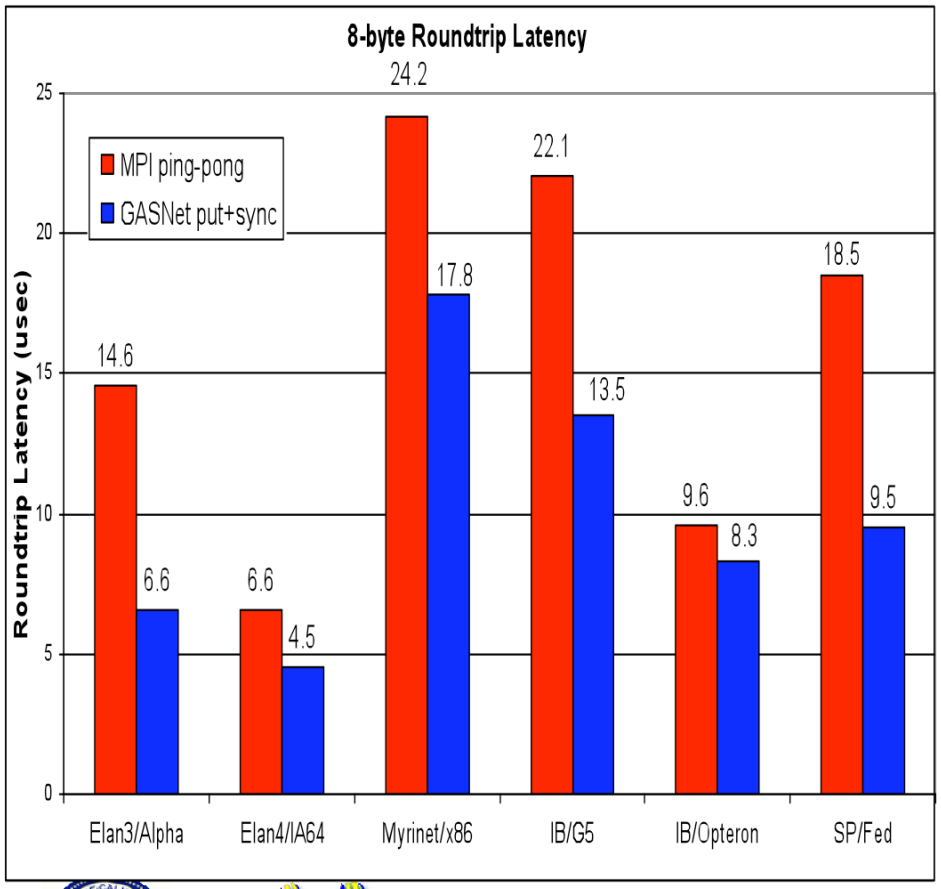
- **UPC LU factorization code adds cooperative (non-preemptive) threads for latency hiding**
  - **New problem in partitioned memory: allocator deadlock**
  - **Can run on of memory locally due tounlucky execution order**

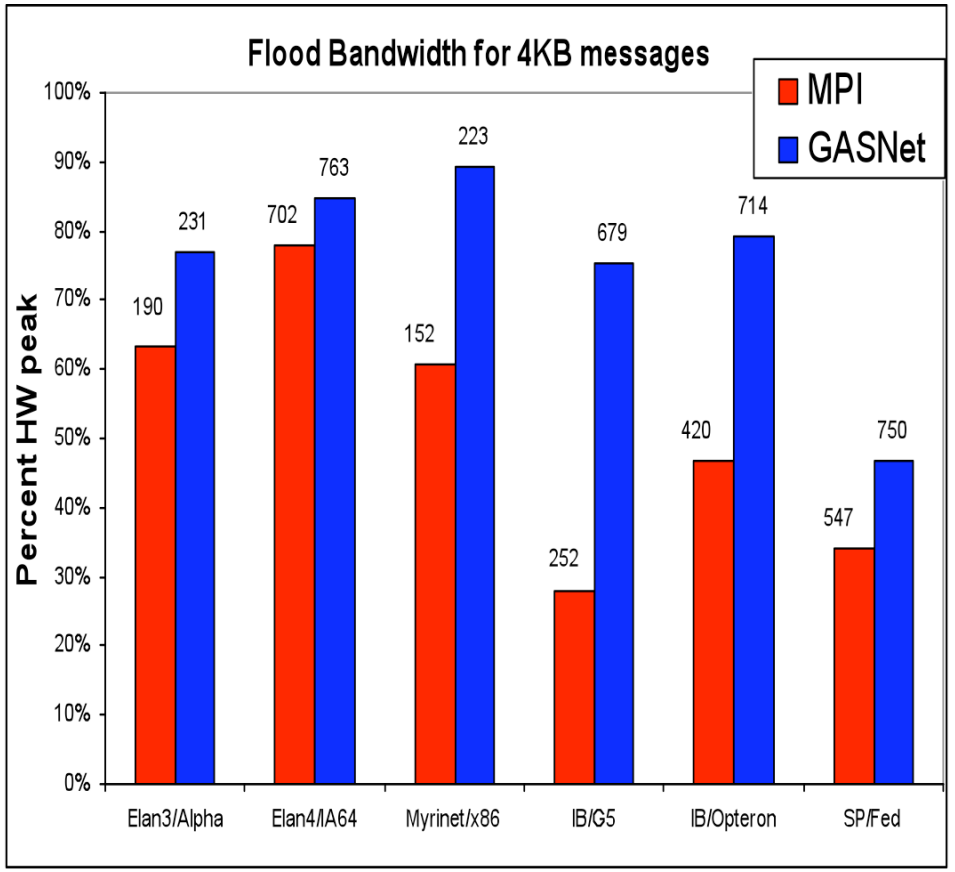


PLASMA by Dongarra et al; UPC LU joint with Parray Husbands

#6: Over Synchronize Communication

- Use a programming model in which you can't utilize bandwidth or "low" latency





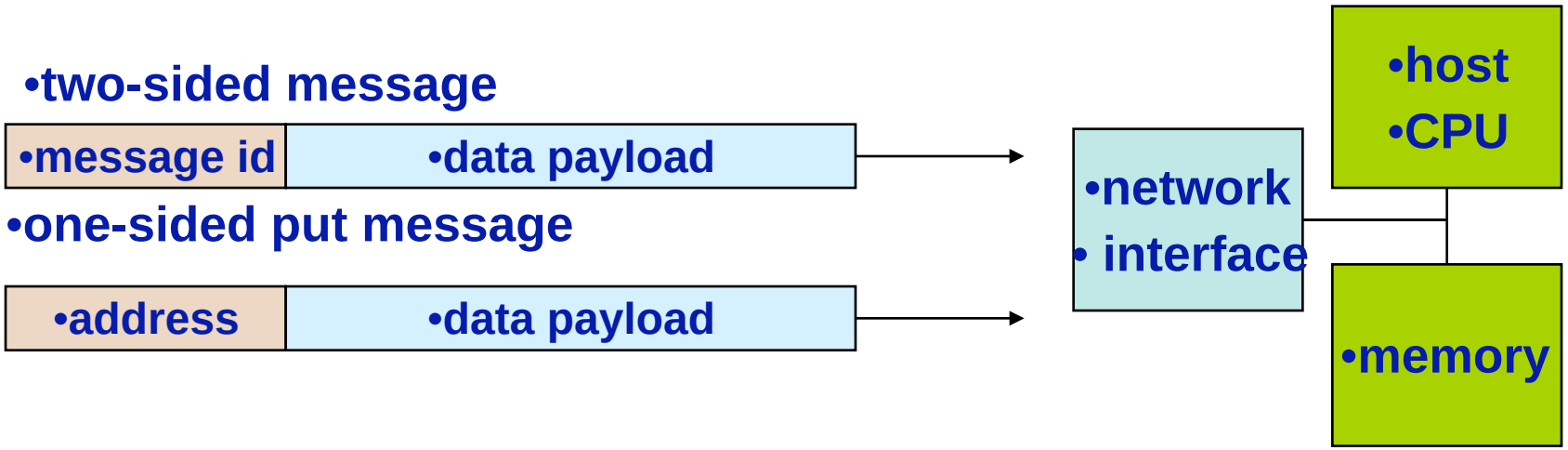
Joint work with Berkeley UPC Group

Berkeley ParLab

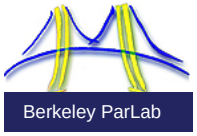
NERSC

Berkeley Lab

Sharing and Communication Models: Two-sided vs One-sided Communication

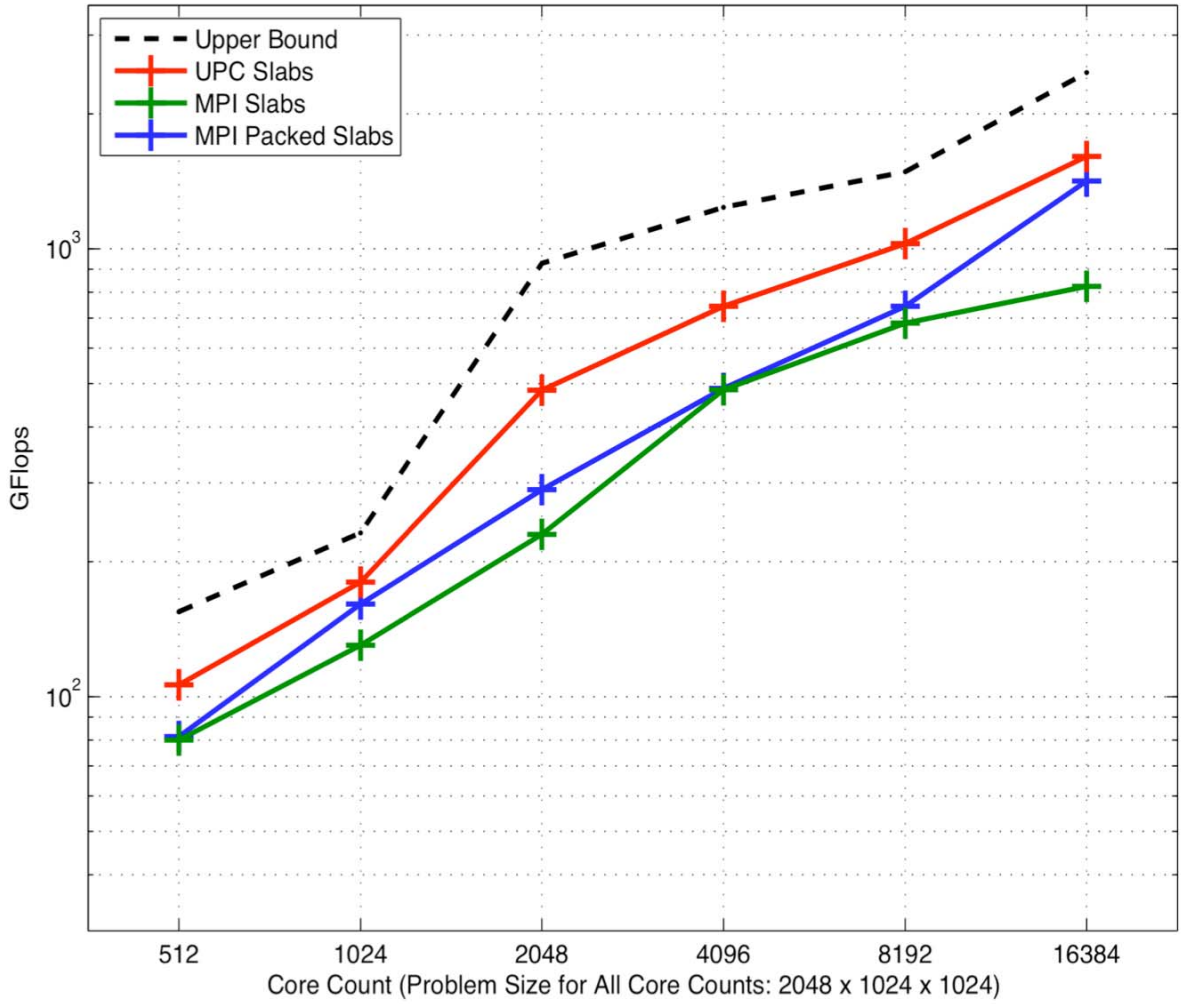


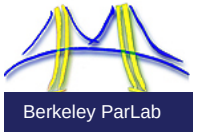
- Two-sided message passing (e.g., MPI) requires matching a send with a receive to identify memory address to put data
  - Wildly popular in HPC, but cumbersome in some applications
  - Couples data transfer with synchronization
- Using global address space decouples synchronization
  - Pay for what you need!
  - Note: Global Addressing ≠ Cache Coherent Shared memory



Joint work with Dan Bonachea, Paul Hargrove, Rajesh Nishtala and rest of UPC group

3D FFT on BlueGene/P

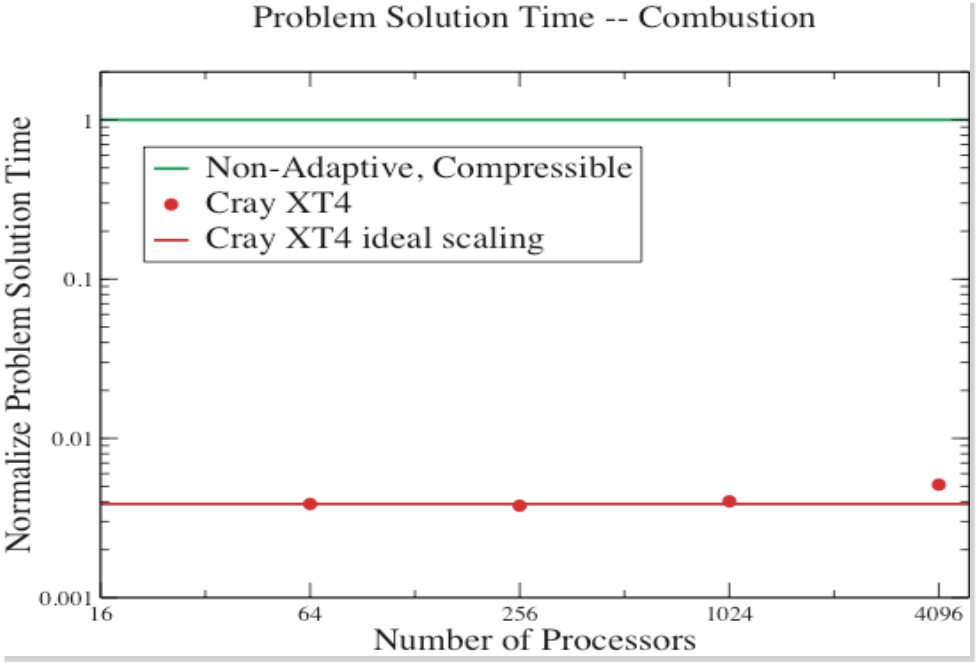




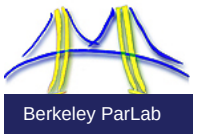
Joint work with Rajesh Nishtala, Dan Bonachea, Paul Hargrove,and rest of UPC group

#7: Run Bad Algorithms

- **Algorithmic gains in last decade have far outstripped Moore's Law**
  - **Adaptive meshes rather than uniform**
  - **Sparse matrices rather than dense**
  - **Reformulation of problem back to basics**



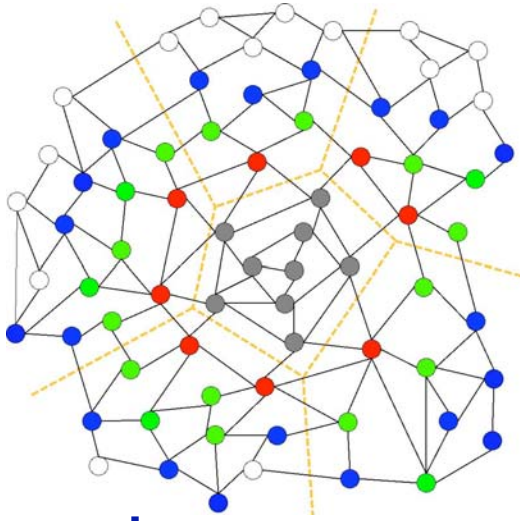
- **Example of canonical "Poisson" problem on n points:**
  - **Dense LU: most general, but $O(n^3)$ flops on $O(n^2)$ data**
  - **Multigrid: fastest/smallest, O(n) flops on O(n) data**

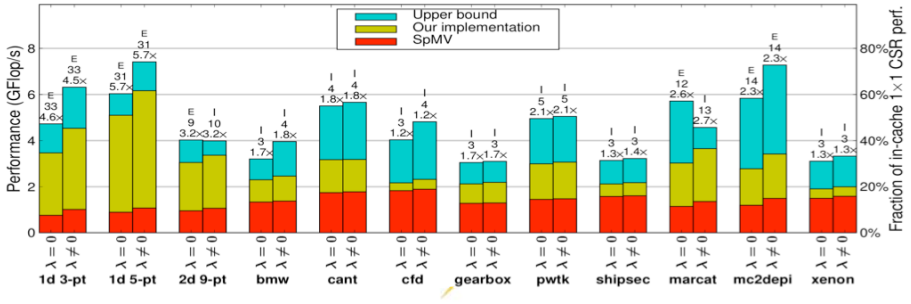


Performance results: John Bell et al

#8: Don't Rethink Your Algorithms

- **Consider Sparse Iterative Methods**
  - Nearest neighbor communication on a mesh
  - Dominated by time to read matrix (edges) from DRAM
  - And (small) communication and global synchronization events at each step
    - **Can we lower data movement costs?**
- Take *k* steps "at once" with one matrix read from DRAM and one communication phase
  - **Parallel implementation**
  - O(log p) messages vs. O(k log p)
  - **Serial implementation**
  - O(1) moves of data moves vs. O(k)
- Performance of $A^k x$ operation relative to Ax and upper boun
  - **Runs up to 5x faster on SMP**

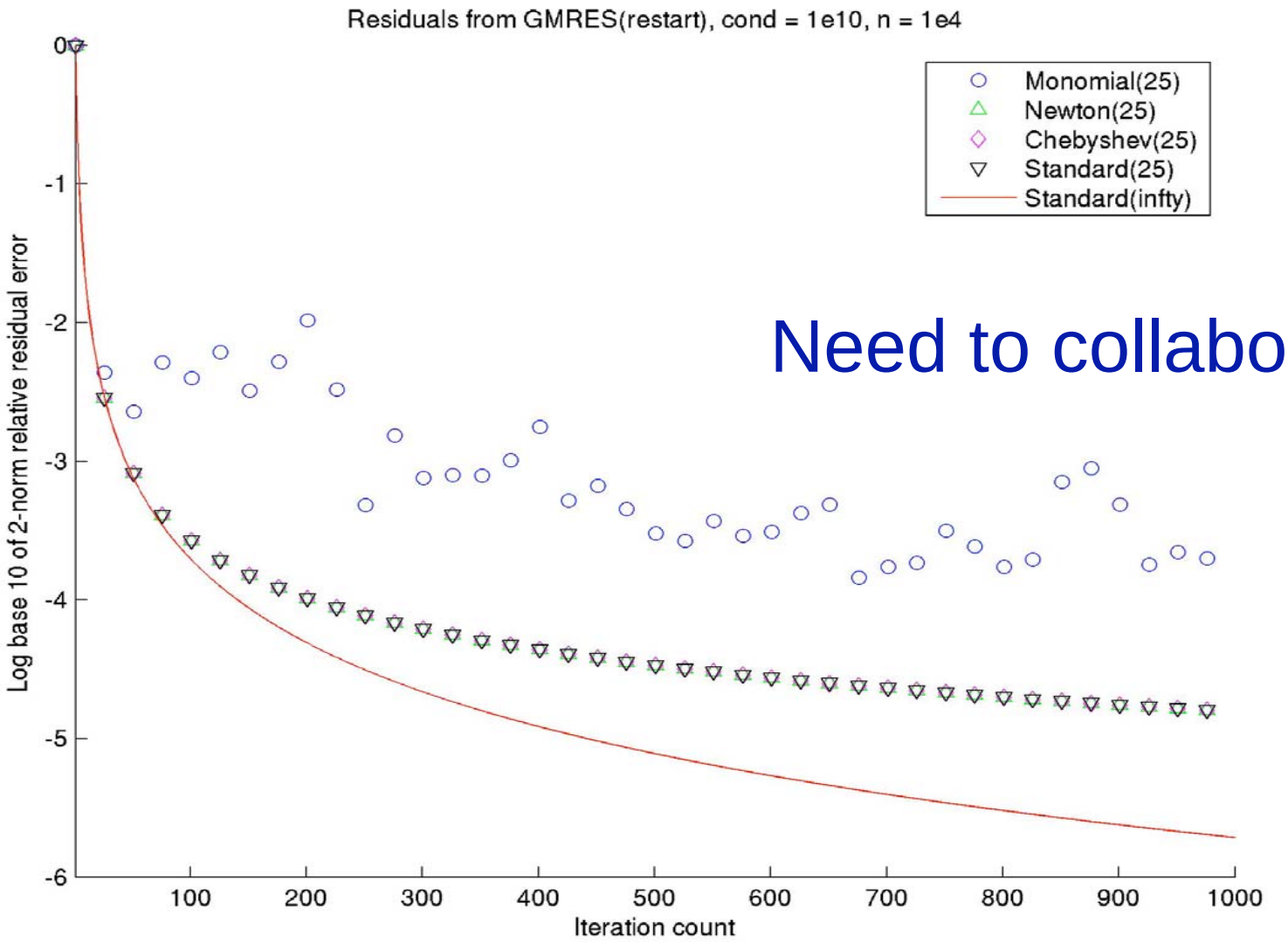




Joint work with Jim Demmel, Mark Hoemman, Marghoob Mohiyuddin

Berkeley ParLab

But the Numerics have to Change!



Need to collaborate

Work by Jim Demmel and Mark Hoemman

Berkeley ParLab

#9: Choose "Hard" Applications ☹

Examples of such systems include

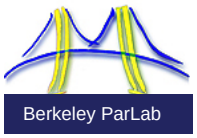
- **Elliptic: steady state, global space dependence**
- **Hyperbolic: time dependent, local space dependence**
- **Parabolic: time dependent, global space dependence**

Global vs Local Dependence

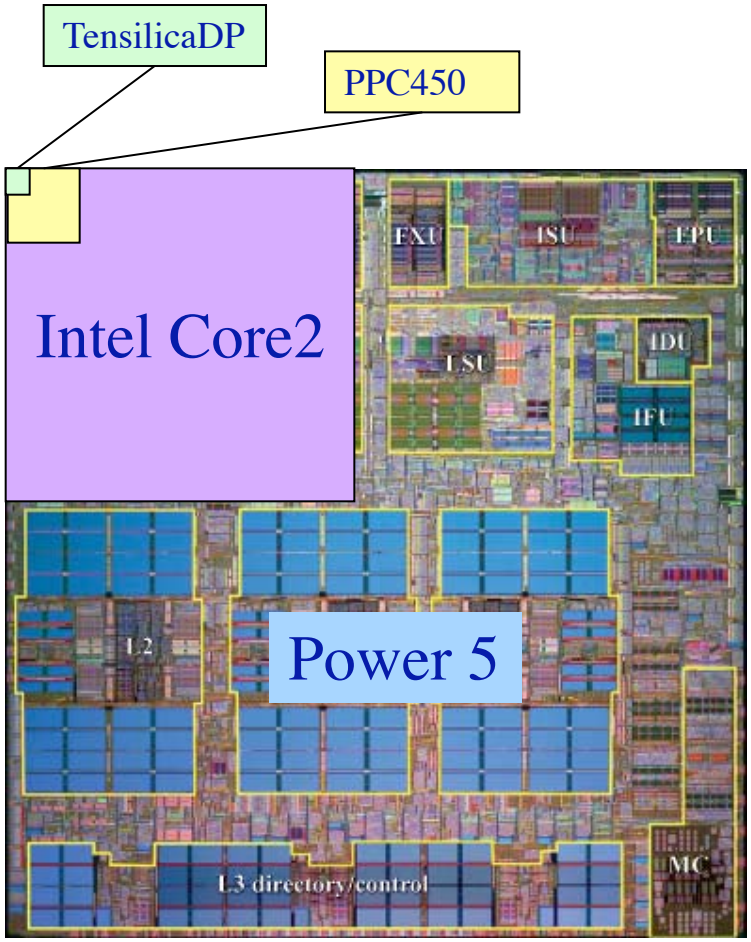
- **Global means either a lot of communication, or tiny time steps: hard to scale well in parallel**
- **Local limits communication, e.g., nearest neighbor**

Global dependencies are inherent in some problems

- **E.g., incompressible fluids like blood flow (games and medicine), ocean dynamics (climate), …**

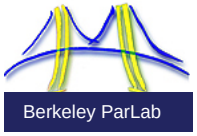


#10: Use Heavy-Weight Cores Optimized for Serial Performance



- **Power5 (Server)**
  - **389 mm²**
  - **120 W @ 1900 MHz**
- **Intel Core2 sc (Laptop)**
  - **130 mm²**
  - **15 W @ 1000 MHz**
- **PowerPC450 (BlueGene/P)**
  - **8 mm²**
  - **3 W @ 850 MHz**
- **Tensilica DP (cell phones)**
  - **0.8 mm²**
  - **0.09 W @ 650 MHz**

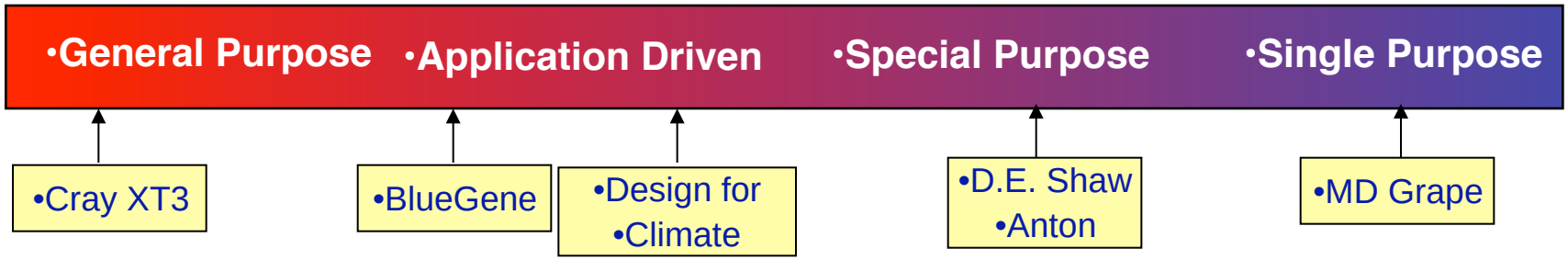
•Each core operates at 1/3 to 1/10th efficiency of largest chip, but you
•can pack 100x more cores onto a chip and consume 1/20 the power!

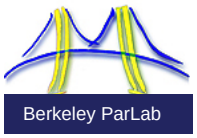


John Shalf and the rest of the Green Flash team

Green Flash Summary

- **We propose a new approach to scientific computing that enables transformational changes for science**
  - **Choose the science target first** *(climate in this case)*
  - **Design systems for applications** *(rather than the reverse)*
  - **Design hardware, software, algorithms together using hardware emulation (*RAMP*) and *auto-tuning***



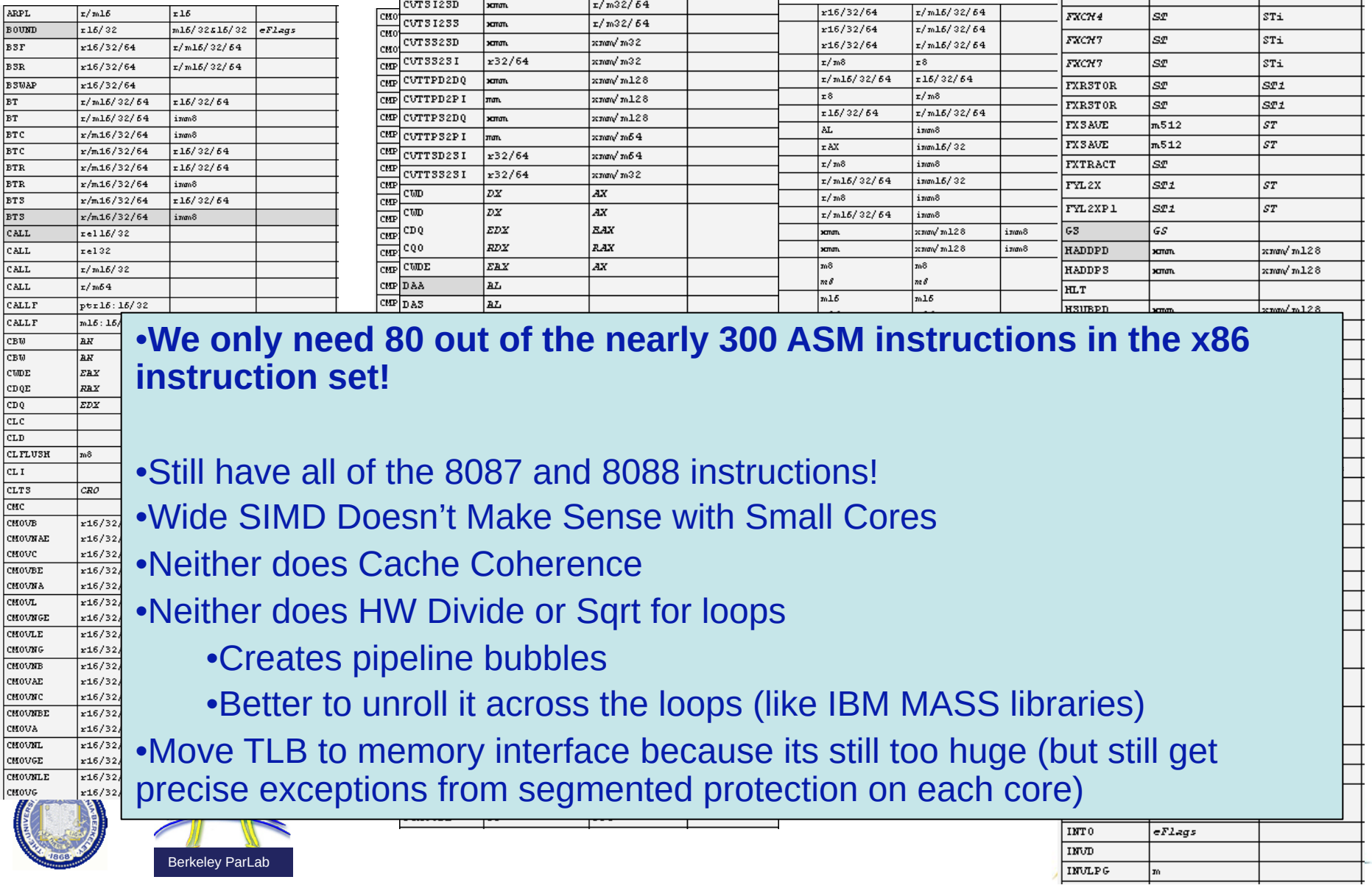


John Shalf and the rest of the Green Flash team

A Short List of x86 Opcodes that Science Applications Don't Need!

mnemonic	op1	op2	op3	op4	iext	pf	0F	po	so	o	proc	st	m	rl	x	tested f	modif f	def f	undef f	f values	description, notes
AAA	AL	AH						37								.....a..	o..szapc	.....a.c	o..sz.p.		ASCII Adjust After Addition
AAD	AL	AH						D5	0A								o..szapc	...sz.p.	o....a.c		ASCII Adjust AX Before Division
AAM	AL	AH						D4	0A								o..szapc	...sz.p.	o....a.c		ASCII Adjust AX After Multiply
AAS	AL	AH						3F								.....a..	o..szapc	.....a.c	o..sz.p.		ASCII Adjust AL After Subtraction
ADC	r/m8	r8						10		r					L	c	o..szapc	o..szapc			Add with Carry
ADC	r/m16/32/64	r16/32/64						11		r					L	c	o..szapc	o..szapc			Add with Carry
ADC	r8	r/m8						12		r						c	o..szapc	o..szapc			Add with Carry
ADC	r16/32/64	r/m16/32/64						13		r						c	o..szapc	o..szapc			Add with Carry
ADC	AL	imm8						14								c	o..szapc	o..szapc			Add with Carry
ADC	rAX	imm16/32						15								c	o..szapc	o..szapc			Add with Carry
ADC	r/m8	imm8						80		2					L	c	o..szapc	o..szapc			Add with Carry
ADC	r/m16/32/64	imm16/32						81		2					L	c	o..szapc	o..szapc			Add with Carry
ADC	r/m8	imm8						82		2					L	c	o..szapc	o..szapc			Add with Carry
ADC	r/m16/32/64	imm8						83		2					L	c	o..szapc	o..szapc			Add with Carry
ADD	r/m8	r8						00		r					L		o..szapc	o..szapc			Add
ADD	r/m16/32/64	r16/32/64						01		r					L		o..szapc	o..szapc			Add
ADD	r8	r/m8						02		r							o..szapc	o..szapc			Add
ADD	r16/32/64	r/m16/32/64						03		r							o..szapc	o..szapc			Add
ADD	AL	imm8						04									o..szapc	o..szapc			Add
ADD	rAX	imm16/32						05									o..szapc	o..szapc			Add
ADD	r/m8	imm8						80		0					L		o..szapc	o..szapc			Add
ADD	r/m16/32/64	imm16/32						81		0					L		o..szapc	o..szapc			Add
ADD	r/m8	imm8						82		0					L		o..szapc	o..szapc			Add
ADD	r/m16/32/64	imm8						83		0					L		o..szapc	o..szapc			Add
ADDPD	xmm	xmm/m128			sse2	66	0F	58		r	P4+										Add Packed Double-FP Values
ADDPS	xmm	xmm/m128			sse1		0F	58		r	P3+										Add Packed Single-FP Values
ADDSD	xmm	xmm/m64			sse2	F2	0F	58		r	P4+										Add Scalar Double-FP Values
ADDSS	xmm	xmm/m32			sse1	F3	0F	58		r	P3+										Add Scalar Single-FP Values
ADDSUBPD	xmm	xmm/m128			sse3	66	0F	D0		r	P4++										Packed Double-FP Add/Subtract
ADDSUBPS	xmm	xmm/m128			sse3	F2	0F	D0		r	P4++										Packed Single-FP Add/Subtract
ADX	AL	AH	imm8					D5									o..szapc	...sz.p.	o....a.c		Adjust AX Before Division
ALTER						64					P4+	v[1]									Alternating branch prefix (used only with Jcc instructions)
AMX	AL	AH	imm8					D4									o..szapc	...sz.p.	o....a.c		Adjust AX After Multiply
AND	r/m8	r8						20		r					L		o..szapc	o..sz.pc	.....a..	o......c	Logical AND
AND	r/m16/32/64	r16/32/64						21		r					L		o..szapc	o..sz.pc	.....a..	o......c	Logical AND
AND	r8	r/m8						22		r							o..szapc	o..sz.pc	.....a..	o......c	Logical AND
AND	r16/32/64	r/m16/32/64						23		r							o..szapc	o..sz.pc	.....a..	o......c	Logical AND
AND	AL	imm8						24									o..szapc	o..sz.pc	.....a..	o......c	Logical AND
AND	rAX	imm16/32						25									o..szapc	o..sz.pc	.....a..	o......c	Logical AND
AND	r/m8	imm8						80		4					L		o..szapc	o..sz.pc	.....a..	o......c	Logical AND
AND	r/m16/32/64	imm16/32						81		4					L		o..szapc	o..sz.pc	.....a..	o......c	Logical AND
AND	r/m8	imm8						82		4					L		o..szapc	o..sz.pc	.....a..	o......c	Logical AND
AND	r/m16/32/64	imm8						83		4	03+				L		o..szapc	o..sz.pc	.....a..	o......c	Logical AND
ANDNPD	xmm	xmm/m128			sse2	66	0F	55		r	P4+										Bitwise Logical AND NOT of Packed Double-FP Values
ANDNPS	xmm	xmm/m128			sse1		0F	55		r	P3+										Bitwise Logical AND NOT of Packed Single-FP Values
ANDPD	xmm	xmm/m128			sse2	66	0F	54		r	P4+										Bitwise Logical AND of Packed Double-FP Values
ANDPS	xmm	xmm/m128			sse1		0F	54		r	P3+										Bitwise Logical AND of Packed Single-FP Values

More Wasted Opcodes



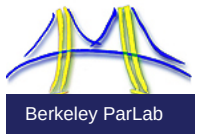
- **We only need 80 out of the nearly 300 ASM instructions in the x86 instruction set!**
- Still have all of the 8087 and 8088 instructions!
- Wide SIMD Doesn't Make Sense with Small Cores
- Neither does Cache Coherence
- Neither does HW Divide or Sqrt for loops
  - Creates pipeline bubbles
  - Better to unroll it across the loops (like IBM MASS libraries)
- Move TLB to memory interface because its still too huge (but still get precise exceptions from segmented protection on each core)

Green Flash Strawman System Design In 2008

We examined three different approaches:

- **AMD Opteron:** Commodity approach, lower efficiency for scientific applications offset by cost efficiencies of mass market
- **BlueGene:** Generic embedded processor core and customize system-on-chip (SoC) services to improve power efficiency for scientific applications
- **Tensilica XTensa:** Customized embedded CPU w/SoC provides further power efficiency benefits but maintains programmability

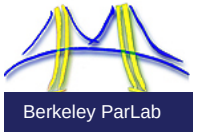
Processor	Clock	Peak/ Core (Gflops)	Cores/ Socket	Sockets	Cores	Power	Cost 2008
AMD Opteron	2.8GHz	5.6	2	890K	1.7M	179 MW	$1B+
IBM BG/P	850MHz	3.4	4	740K	3.0M	20 MW	$1B+
Green Flash / Tensilica XTensa	650MHz	2.7	32	120K	4.0M	3 MW	$75M



John Shalf and the rest of the Green Flash team

Ten Sources of Waste in Parallel Computing

1) Insufficient memory bandwidth (HW)
2) Ignore performance features (SW+HW)
3) Ignore Little's Law (SW+HW)
4) Hide faults in low level (SW+HW)
5) Over synchronization globally (SW)
6) Over synchronize communication (SW)
7) Choose bad algorithms (Alg)
8) Don’t rethink algorithms (Alg)
9) Choose “hard” applications (Apps)
10) Use overly-general processors (HW)



Conclusions

- **Enable programmers to get performance**
  - **Expose features for performance**
  - **Don't hide them**
- **Go Green**
  - **Enable energy-efficient computers and software**
- **Work with experts on software, algorithms, applications**