Green Flash: Application Driven System Design for Power Efficient HPC

John Shalf

David Donofrio, Leonid Oliker, Michael Wehner

And many other CRD and NERSC staff

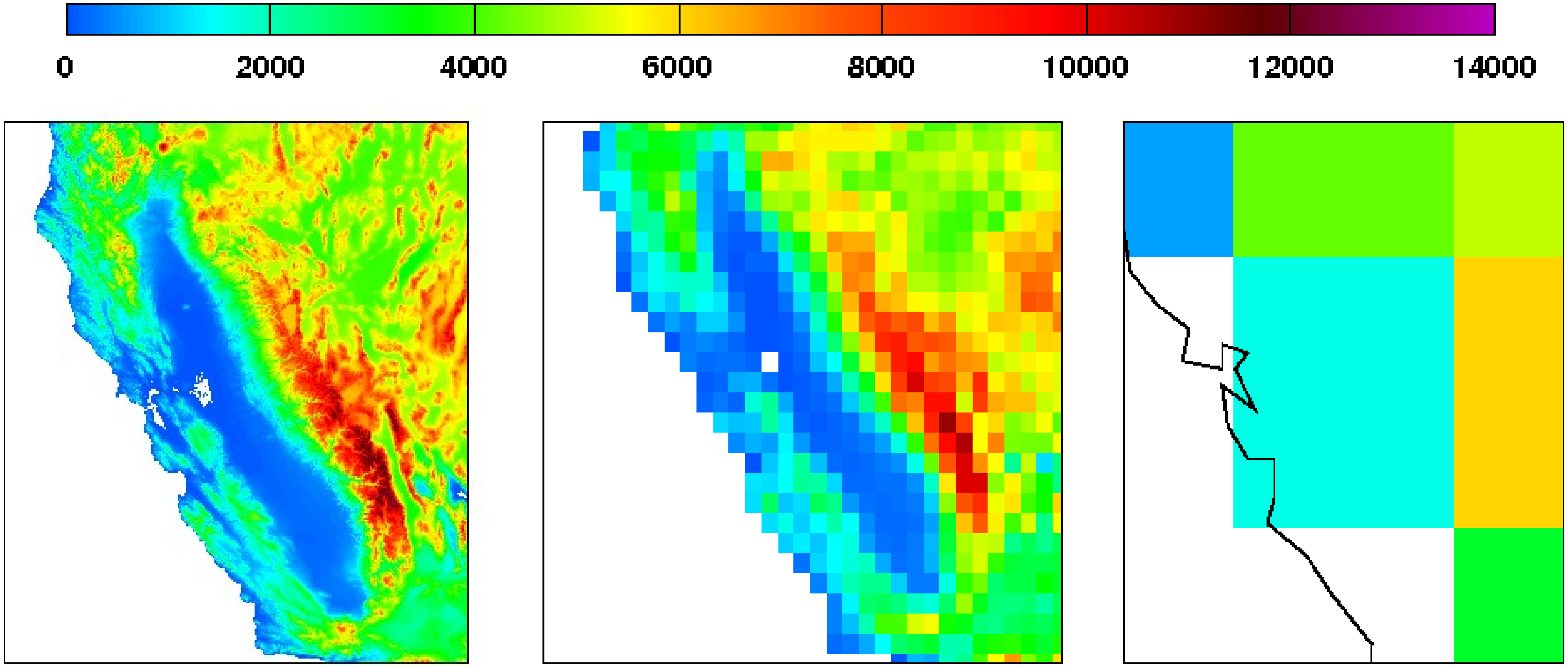
Salishan, April 2009

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, scientific algorithms together using hardware emulation (*RAMP*) and *auto-tuning***
  - **This is the right way to design efficient HPC systems!**

Apply approach to broad range of Exascale-class scientific applications

Global Cloud System Resolving Models are a Transformational Change



1km
Cloud system resolving models

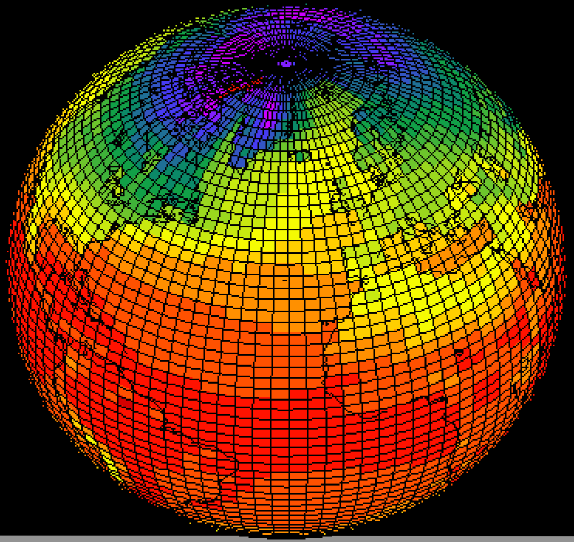
25km
Upper limit of climate models with cloud parameterizations

200km
Typical resolution of IPCC AR4 models

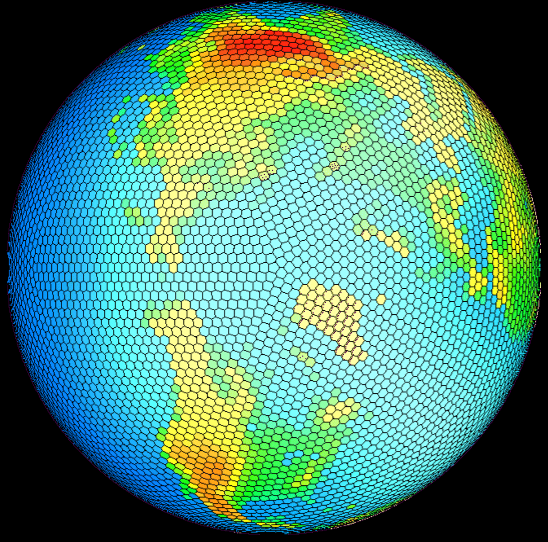
Requirements for 1km Climate Computer

Must maintain 1000x faster than real time for practical climate simulation

- ~2 million horizontal subdomains
- 100 Terabytes of Memory
  - 5MB memory per subdomain
- ~20 million total subdomains
  - 20 PF sustained (200PF peak)
  - Nearest-neighbor communication
- *New discretization for climate model*
  - *CSU Icosahedral Code*

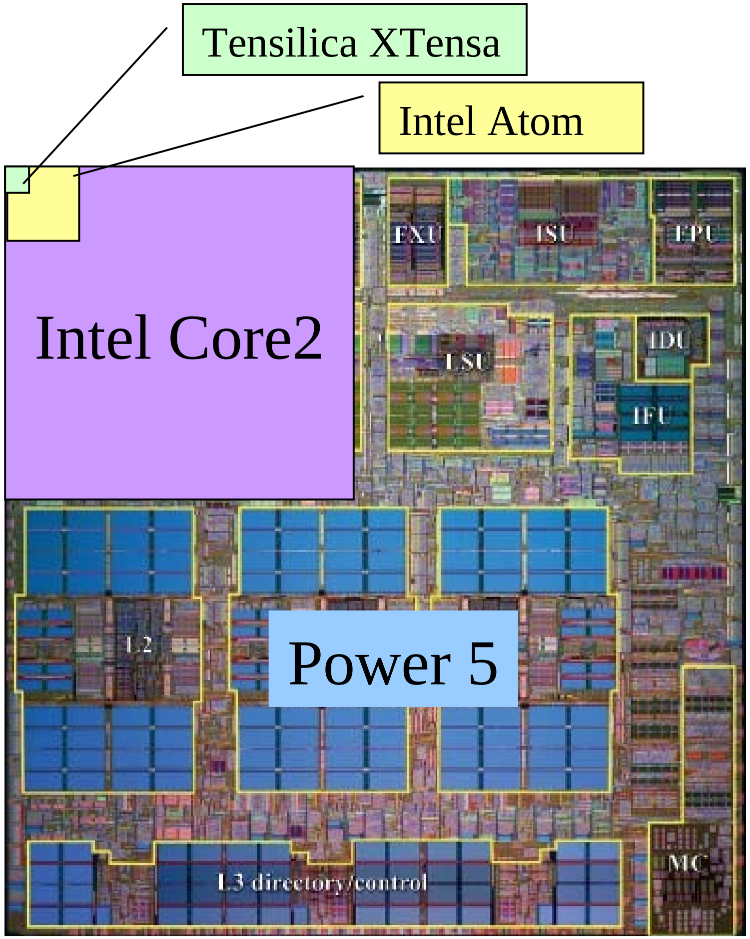


fvCAM



Icosahedral

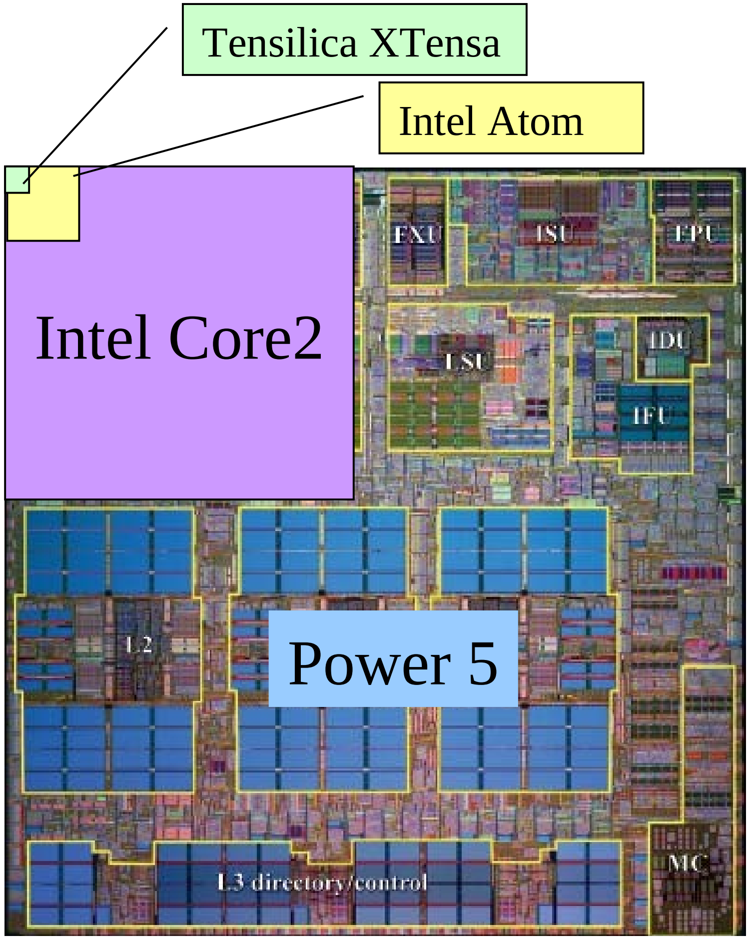
Low-Power Design Principles



- **Cubic power improvement with lower clock rate due to V^2F**
- **Slower clock rates enable use of simpler cores**
- **Simpler cores use less area (lower leakage) and reduce cost**
- **Tailor design to application to REDUCE WASTE**

This is how iPhones and MP3 players are designed to maximize battery life and minimize cost

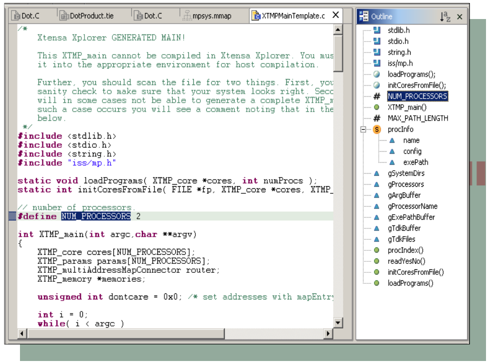
Low-Power Design Principles



- **Power5 (server)**
  - **120W@1900MHz**
  - **Baseline**
- **Intel Core2 sc (laptop) :**
  - **15W@1000MHz**
  - ***4x more FLOPs/watt than baseline***
- **Intel Atom (handhelds)**
  - **0.625W@800MHz**
  - **80x more**
- **Tensilica XTensa (Moto Razor) :**
  - **0.09W@600MHz**
  - **400x more *(80x-120x sustained)***

Embedded Design Automation

(Example from Existing Tensilica Design Flow)

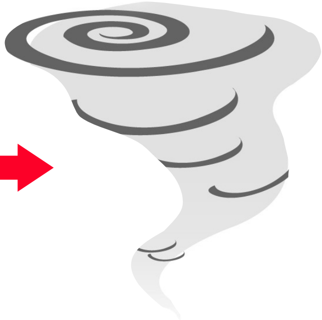


```
/*
    Xtensa Xplorer GENERATED MAIN!

    This XTMP_main cannot be compiled in Xtensa Xplorer. You mus
    it into the appropriate environment for host compilation.

    Further, you should scan the file for two things. First, you
    sanity check to make sure that your system looks right. Secc
    will in some cases not be able to generate a complete XTMP_m
    such a case occurs you will see a comment noting that in the
    below.
 */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "iss/mp.h"

static void loadPrograms( XTMP_core *cores, int numProcs );
static int initCoresFromFile( FILE *fp, XTMP_core *cores, XTMP_

// number of processors.
#define NUM_PROCESSORS 2

int XTMP_main(int argc,char **argv)
{
    XTMP_core cores[NUM_PROCESSORS];
    XTMP_params params[NUM_PROCESSORS];
    XTMP_multiAddressMapConnector router;
    XTMP_memory *memories;

    unsigned int dontcare = 0x0; /* set addresses with mapEntr

    int i = 0;
    while( i < argc )
```

```
stdlib.h
stdio.h
string.h
iss/mp.h
loadPrograms();
initCoresFromFile();
NUM_PROCESSORS
XTMP_main()
MAX_PATH_LENGTH
procInfo
    name
    config
    exePath
gSystemDirs
gProcessors
gArgBuffer
gProcessorName
gExePathBuffer
gTdkBuffer
gTdkFiles
procIndex()
readYesNo()
initCoresFromFile()
loadPrograms()
```

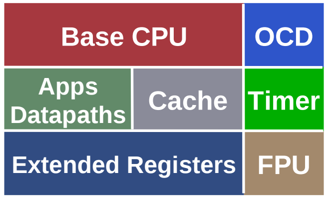
Processor configuration

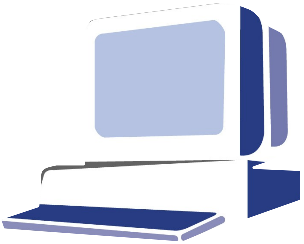
1. Select from menu
2. Automatic instruction discovery (XPRES Compiler)
3. Explicit instruction description (TIE)



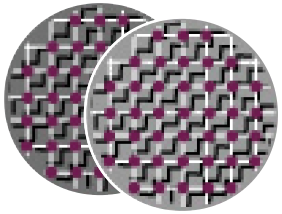
Processor Generator (*Tensilica*)

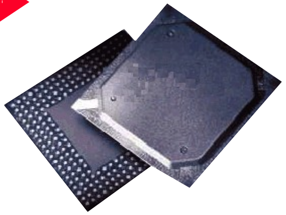
Application-optimized processor implementation (RTL/Verilog)





Tailored SW Tools: Compiler, debugger, simulators, Linux, other OS Ports *(Automatically generated together with the Core)*



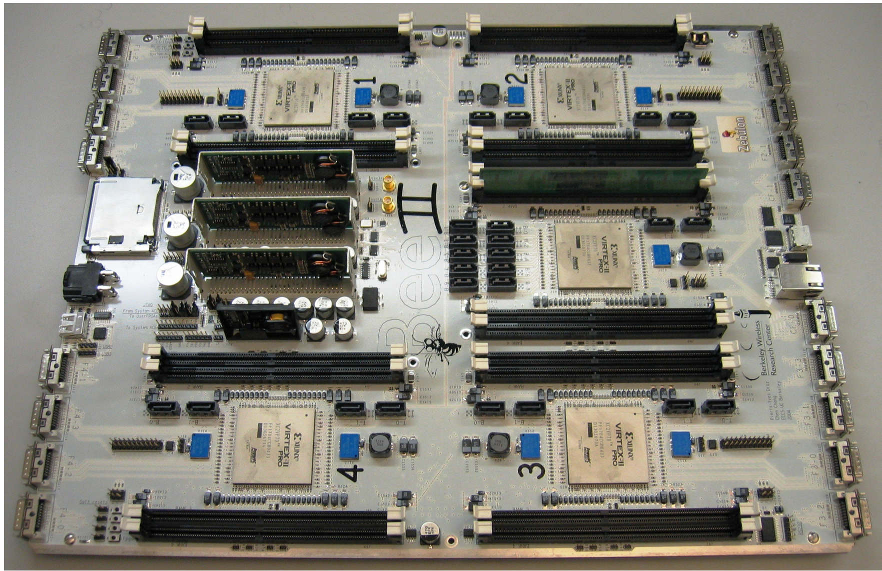


Build with any process in any fab

Advanced Hardware Simulation (RAMP)

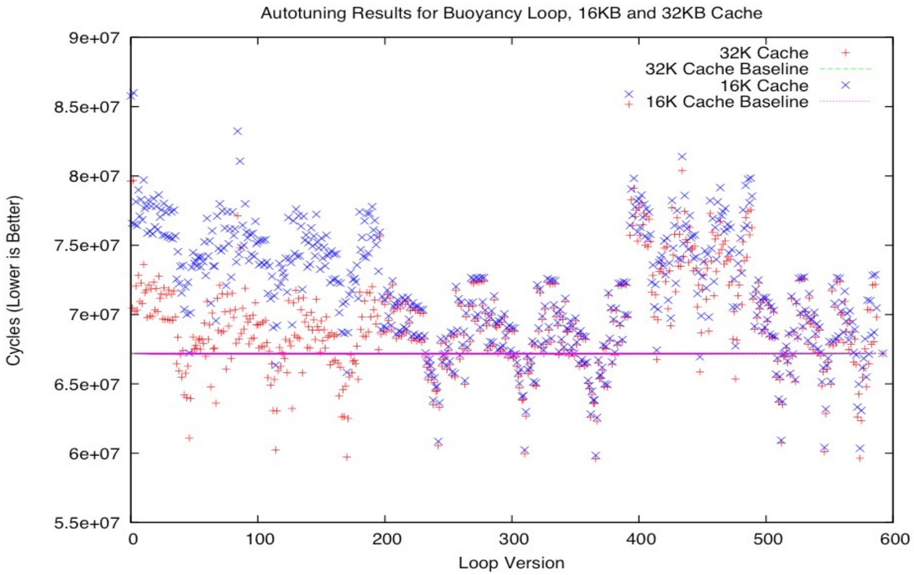
Enabling Hardware/Software/Science Co-Design

- **Research Accelerator for Multi-Processors (RAMP)**
  - **Simulate hardware *before* it is built!**
  - **Break slow feedback loop for system designs**
  - **Enables tightly coupled hardware/software/science co-design** *(not possible using conventional approach)*

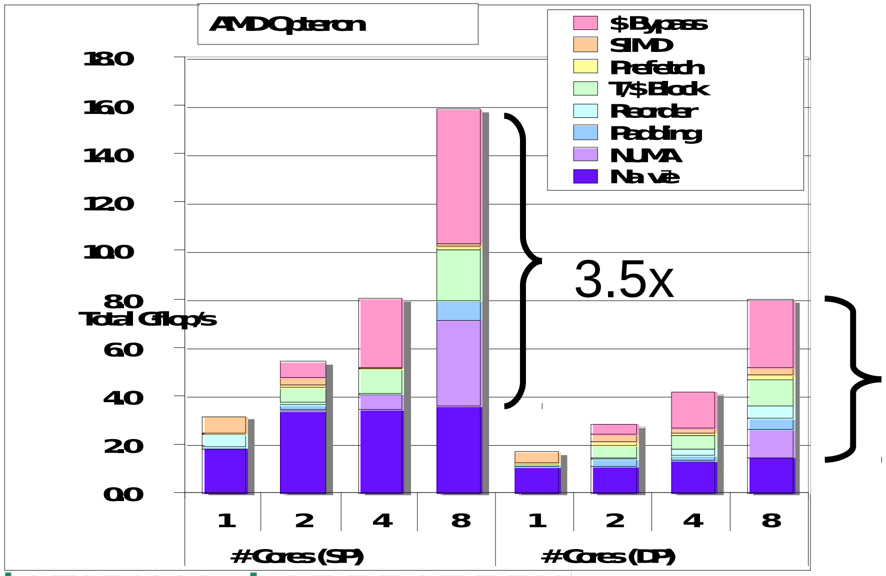


Auto-tuning

- **Problem: want to compare best potential performance of diverse architectures, avoiding**
  - Non-portable code
  - Labor-intensive user optimizations for each specific architecture



- **Our Solution: Auto-tuning**
  - **Automate search across a complex optimization space**
  - **Achieve performance far beyond current compilers**
  - **achieve performance portability for diverse architectures!**

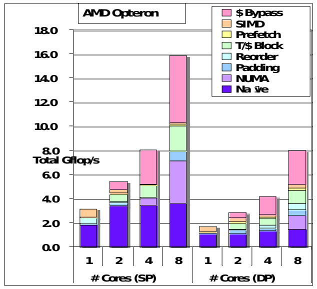


Traditional New Architecture Hardware/Software Design

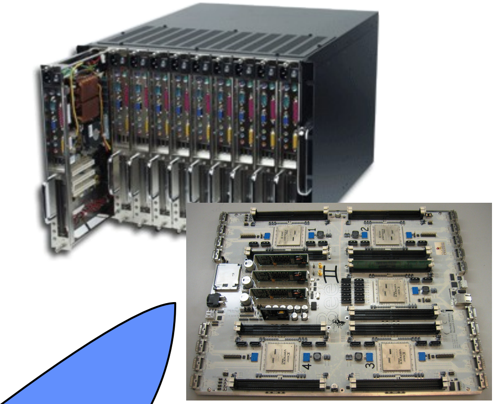
How long does it take for a full scale application to influence architectures?

Design New System
(2 year concept phase)

Tune Software
(2 years)



Cycle Time
4-6+ years



Build Hardware
(2 years)

Port Application

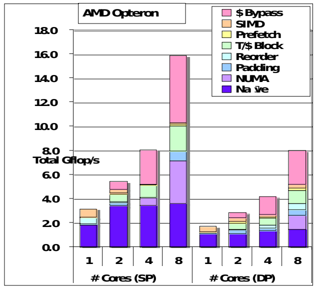
Proposed New Architecture Hardware/Software Co-Design

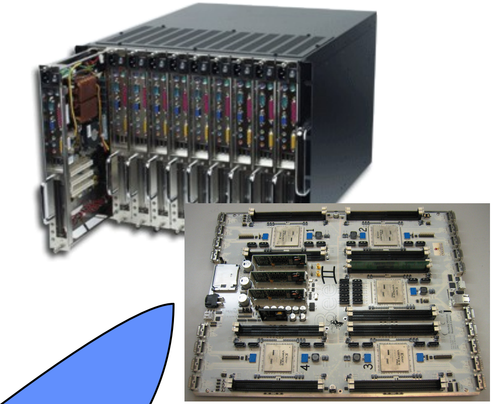
How long does it take for a full scale application to influence architectures?

Synthesize SoC (hours)

Cycle Time
1-2 days

Autotune
Software
(Hours)



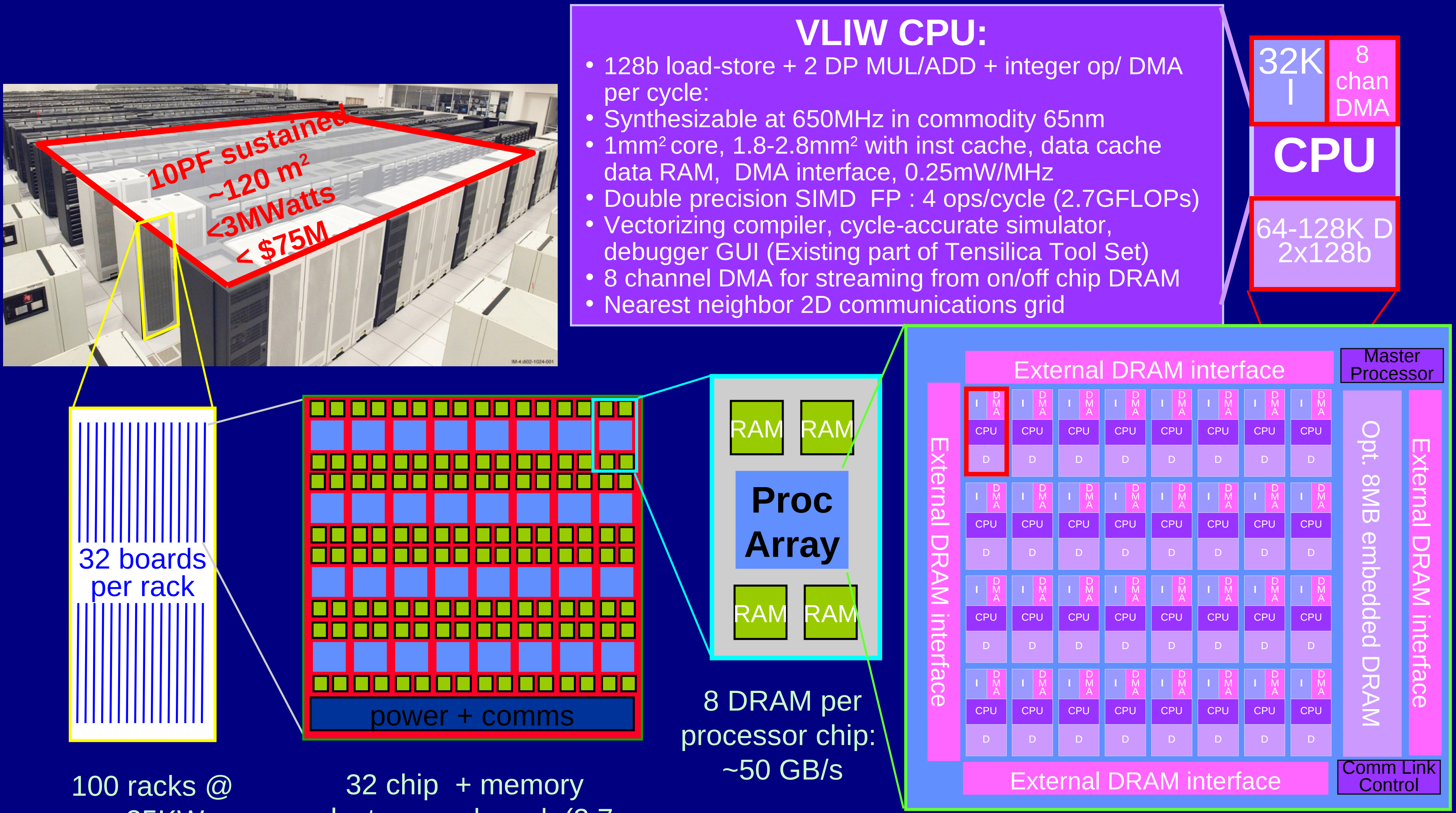


Emulate
Hardware
(RAMP)
(hours)

Build application

Climate System Design Concept

Strawman Design Study



VLIW CPU:

- 128b load-store + 2 DP MUL/ADD + integer op/ DMA per cycle:
- Synthesizable at 650MHz in commodity 65nm
- 1mm^2 core, 1.8-2.8mm^2 with inst cache, data cache data RAM, DMA interface, 0.25mW/MHz
- Double precision SIMD FP : 4 ops/cycle (2.7GFLOPs)
- Vectorizing compiler, cycle-accurate simulator, debugger GUI (Existing part of Tensilica Tool Set)
- 8 channel DMA for streaming from on/off chip DRAM
- Nearest neighbor 2D communications grid

100 racks @ ~25KW

32 chip + memory clusters per board (2.7 TFLOPS @ 700W

8 DRAM per processor chip: ~50 GB/s

32 processors per 65nm chip 83 GFLOPS @ 7W

Green Flash Strawman System Design In 2008

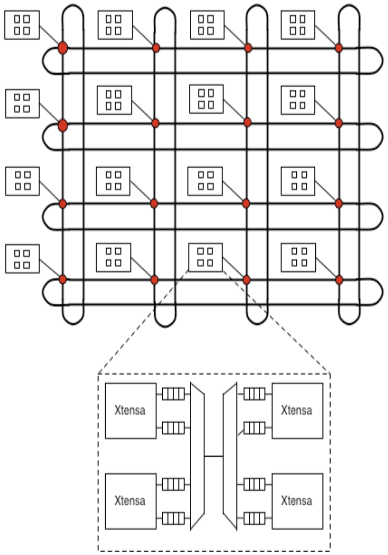
We examined three different approaches:

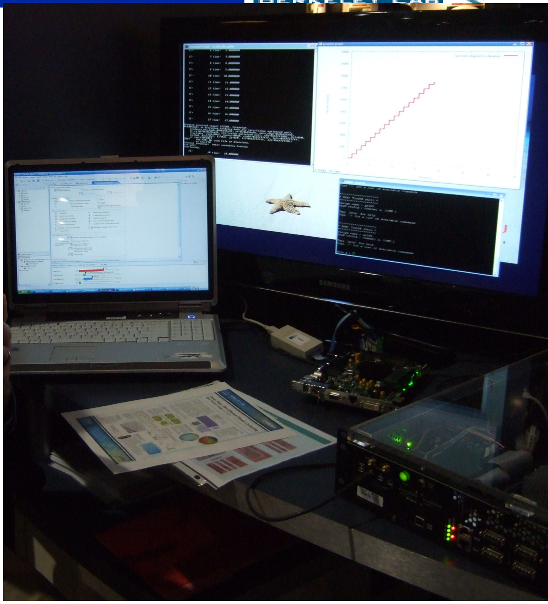
- **AMD Opteron:** Commodity approach, lower efficiency for scientific applications offset by cost efficiencies of mass market
- **BlueGene:** Generic embedded processor core and customize system-on-chip (SoC) services to improve power efficiency for scientific applications
- **Tensilica XTensa:** Customized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

Green Flash Hardware Demo

- Demonstrated during SC ‘08
- Proof of concept
  - CSU atmospheric model ported to Tensilica Architecture
  - Single Tensilica processor running atmospheric model at 50MHz
- Emulation performance advantage
  - Processor running at 50MHz vs. Functional model at 100 kHz
  - 500x Speedup
- Actual code running - not representative benchmark



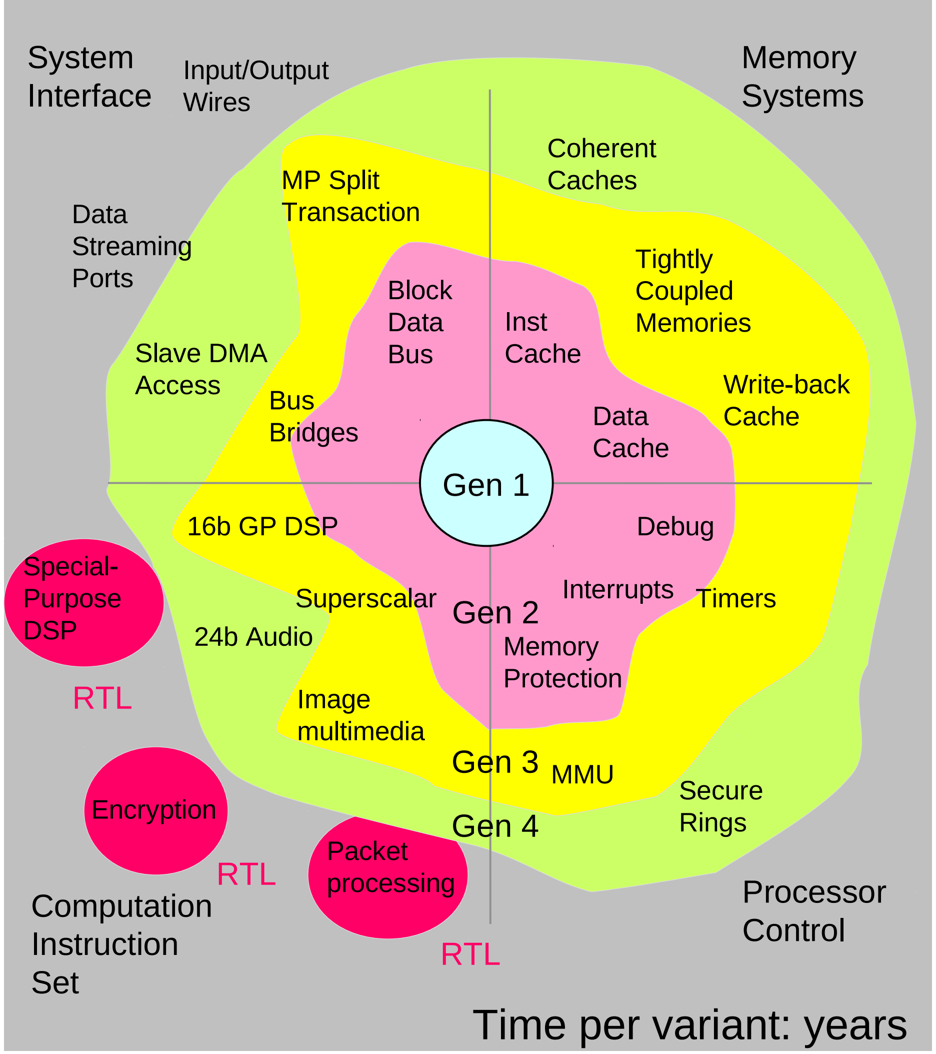


What Have We Learned?

Peel Back the Historical Growth of Instruction Sets *(accretion of cruft)*

From Chris Rowen (Tensilica Inc)

Traditional Processor Family



System Interface

Input/Output Wires

Memory Systems

Coherent Caches

MP Split Transaction

Data Streaming Ports

Block Data Bus

Inst Cache

Tightly Coupled Memories

Slave DMA Access

Bus Bridges

Write-back Cache

Data Cache

Gen 1

16b GP DSP

Debug

Special-Purpose DSP

Superscalar

Gen 2

Interrupts

Timers

24b Audio

Memory Protection

RTL

Image multimedia

Gen 3

MMU

Encryption

Secure Rings

Gen 4

RTL

Packet processing

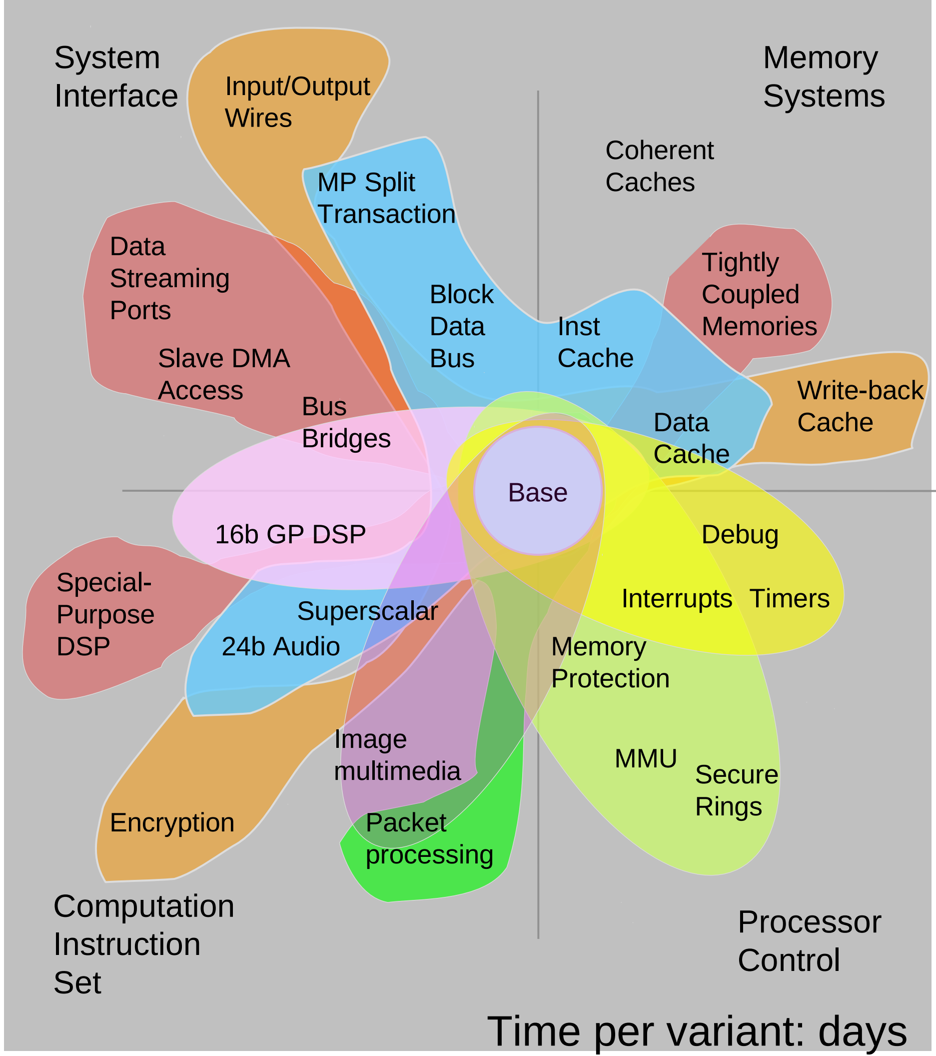
Computation Instruction Set

Processor Control

RTL

Time per variant: years

Configurable Processor Family



System Interface

Input/Output Wires

Memory Systems

Coherent Caches

MP Split Transaction

Data Streaming Ports

Block Data Bus

Inst Cache

Tightly Coupled Memories

Slave DMA Access

Bus Bridges

Write-back Cache

Data Cache

Base

16b GP DSP

Debug

Special-Purpose DSP

Superscalar

Interrupts

Timers

24b Audio

Memory Protection

Image multimedia

MMU

Secure Rings

Encryption

Packet processing

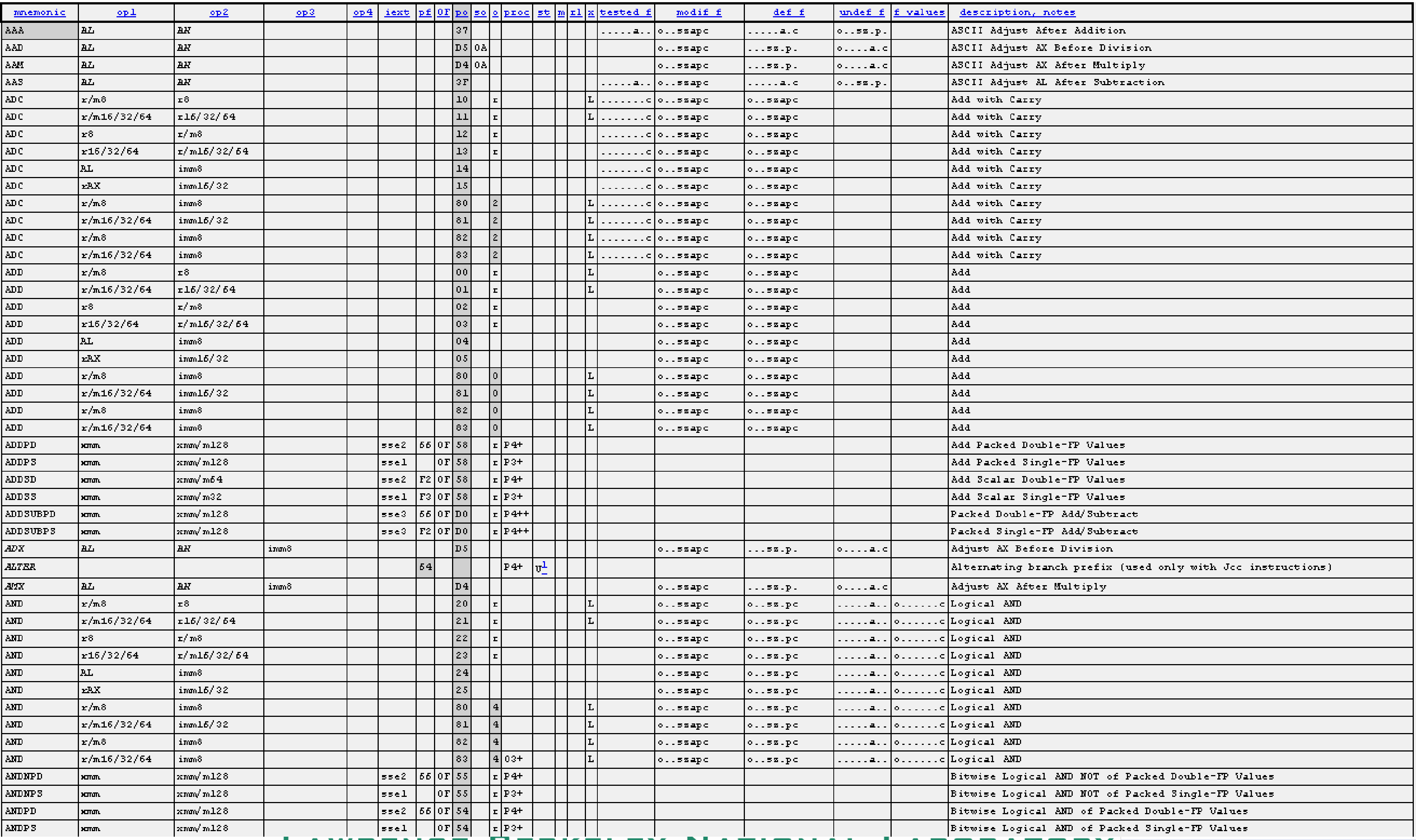
Computation Instruction Set

Processor Control

Time per variant: days

Area = silicon cost and power

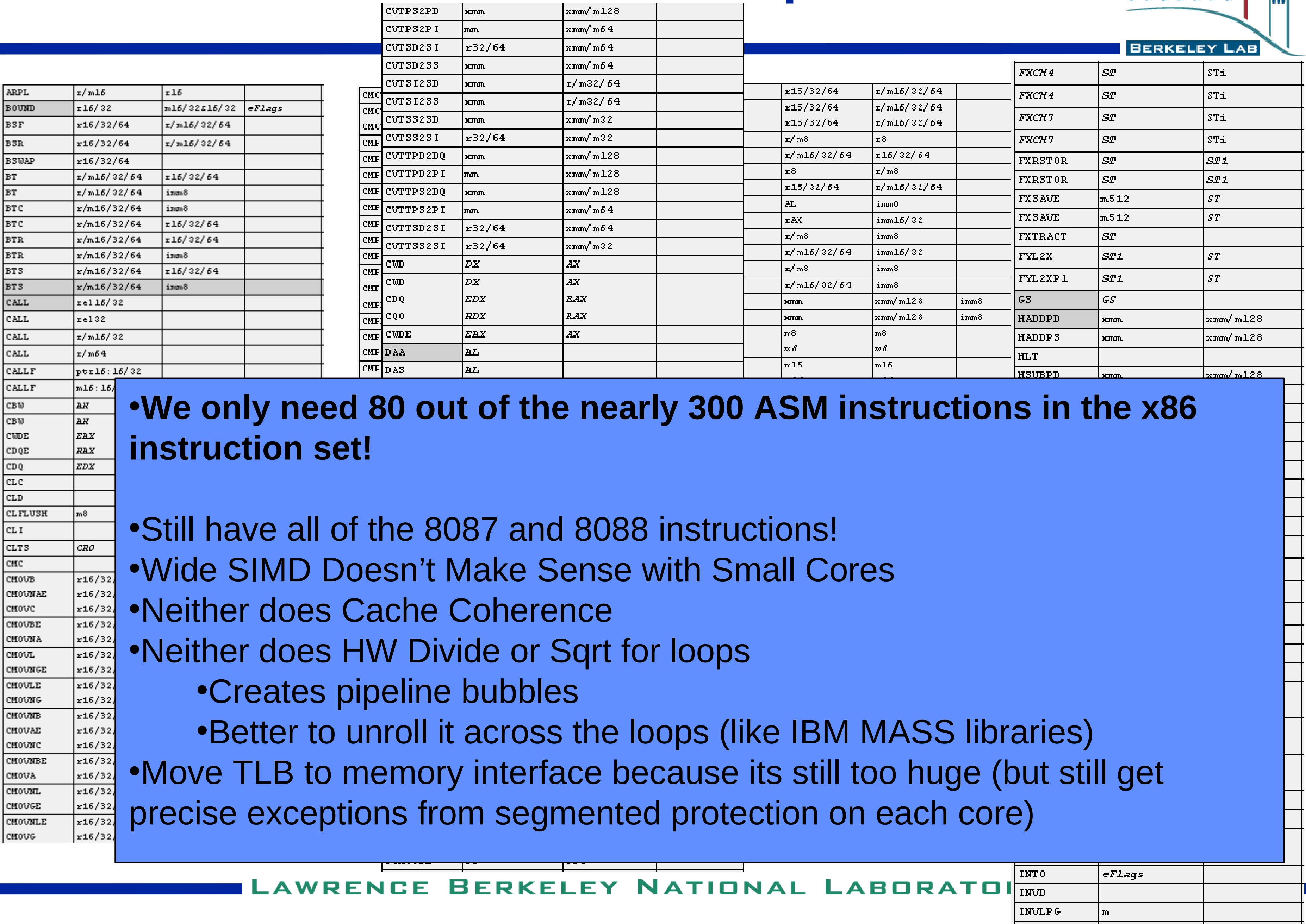
A Short List of x86 Opcodes that Science Applications Don’t Need!



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+	U[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



ARPL	r/m16	r16	
BOUND	r16/32	m16/32&16/32	*eFlags*
BSF	r16/32/64	r/m16/32/64	
BSR	r16/32/64	r/m16/32/64	
BSWAP	r16/32/64		
BT	r/m16/32/64	r16/32/64	
BT	r/m16/32/64	imm8	
BTC	r/m16/32/64	imm8	
BTC	r/m16/32/64	r16/32/64	
BTR	r/m16/32/64	r16/32/64	
BTR	r/m16/32/64	imm8	
BTS	r/m16/32/64	r16/32/64	
BTS	r/m16/32/64	imm8	
CALL	rel16/32		
CALL	rel32		
CALL	r/m16/32		
CALL	r/m64		
CALLF	ptr16:16/32		
CALLF	m16:16/		
CBW	*AH*		
CBW	*AH*		
CWDE	*EAX*		
CDQE	*RAX*		
CDQ	*EDX*		
CLC			
CLD			
CLFLUSH	m8		
CLI			
CLTS	*CR0*		
CMC			
CMOVB	r16/32/		
CMOVNAE	r16/32/		
CMOVC	r16/32/		
CMOVBE	r16/32/		
CMOVNA	r16/32/		
CMOVL	r16/32/		
CMOVNGE	r16/32/		
CMOVLE	r16/32/		
CMOVNG	r16/32/		
CMOVNB	r16/32/		
CMOVAE	r16/32/		
CMOVNC	r16/32/		
CMOVNBE	r16/32/		
CMOVA	r16/32/		
CMOVNL	r16/32/		
CMOVGE	r16/32/		
CMOVNLE	r16/32/		
CMOVG	r16/32/		

CMO	r16/32/64	r/m16/32/64	
CMO	r16/32/64	r/m16/32/64	
CMO	r16/32/64	r/m16/32/64	
CMP	r/m8	r8	
CMP	r/m16/32/64	r16/32/64	
CMP	r8	r/m8	
CMP	r16/32/64	r/m16/32/64	
CMP	AL	imm8	
CMP	rAX	imm16/32	
CMP	r/m8	imm8	
CMP	r/m16/32/64	imm16/32	
CMP	r/m8	imm8	
CMP	r/m16/32/64	imm8	
CMP	xmm	xmm/m128	imm8
CMP	xmm	xmm/m128	imm8
CMP	m8	m8	
CMP	*m8*	*m8*	
CMP	m16	m16	

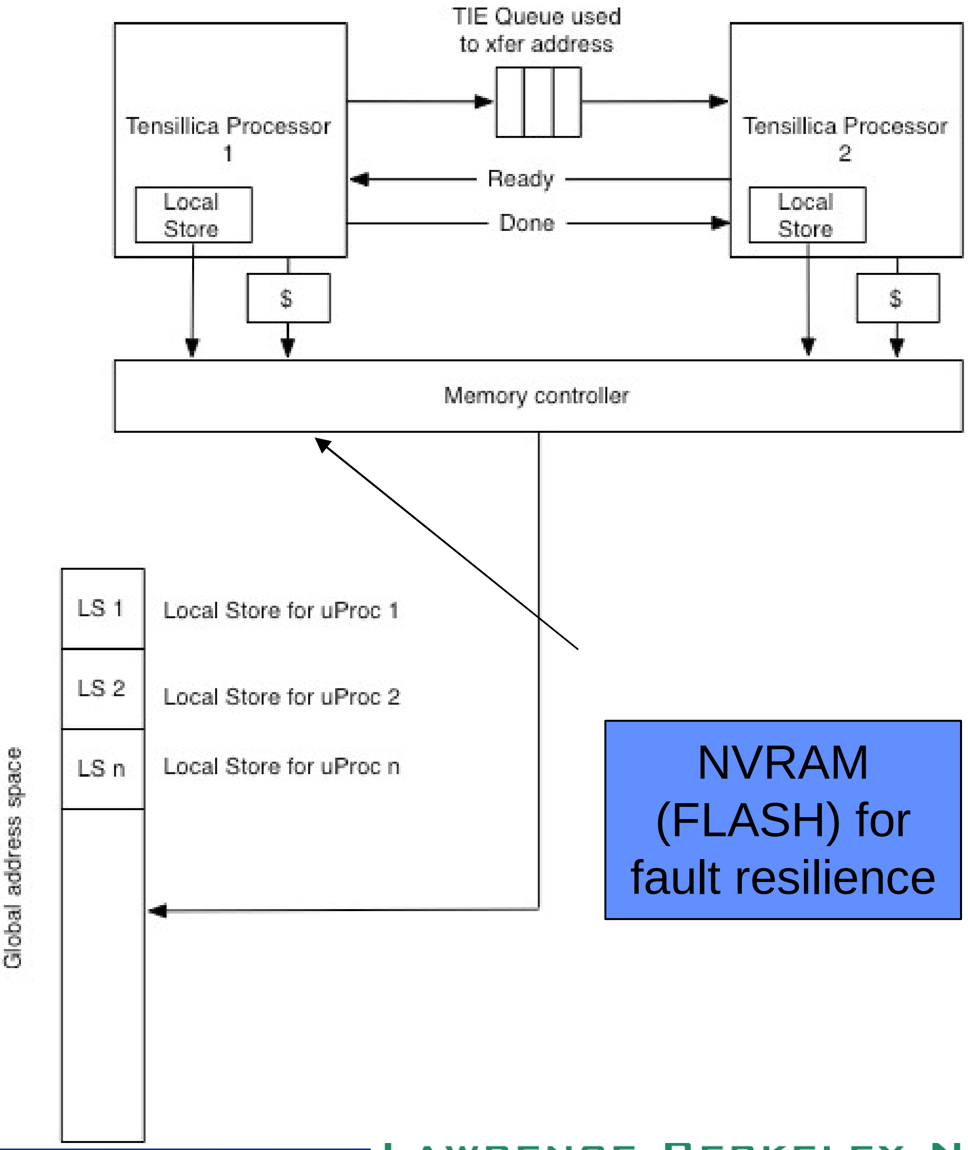
CVTPS2PD	xmm	xmm/m128	
CVTPS2PI	mm	xmm/m64	
CVTSD2SI	r32/64	xmm/m64	
CVTSD2SS	xmm	xmm/m64	
CVTSI2SD	xmm	r/m32/64	
CVTSI2SS	xmm	r/m32/64	
CVTSS2SD	xmm	xmm/m32	
CVTSS2SI	r32/64	xmm/m32	
CVTTPD2DQ	xmm	xmm/m128	
CVTTPD2PI	mm	xmm/m128	
CVTTPS2DQ	xmm	xmm/m128	
CVTTPS2PI	mm	xmm/m64	
CVTTSD2SI	r32/64	xmm/m64	
CVTTSS2SI	r32/64	xmm/m32	
CWD	*DX*	*AX*	
CWD	*DX*	*AX*	
CDQ	*EDX*	*EAX*	
CQO	*RDX*	*RAX*	
CWDE	*EAX*	*AX*	
DAA	*AL*		
DAS	*AL*		

FXCH4	*ST*	STi
FXCH4	*ST*	STi
FXCH7	*ST*	STi
FXCH7	*ST*	STi
FXRSTOR	*ST*	*ST1*
FXRSTOR	*ST*	*ST1*
FXSAVE	m512	*ST*
FXSAVE	m512	*ST*
FXTRACT	*ST*	
FYL2X	*ST1*	*ST*
FYL2XP1	*ST1*	*ST*
GS	*GS*	
HADDPD	xmm	xmm/m128
HADDPS	xmm	xmm/m128
HLT		
HSUBPD	xmm	xmm/m128
INTO	*eFlags*	
INVD		
INVLPG	m	

- **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)

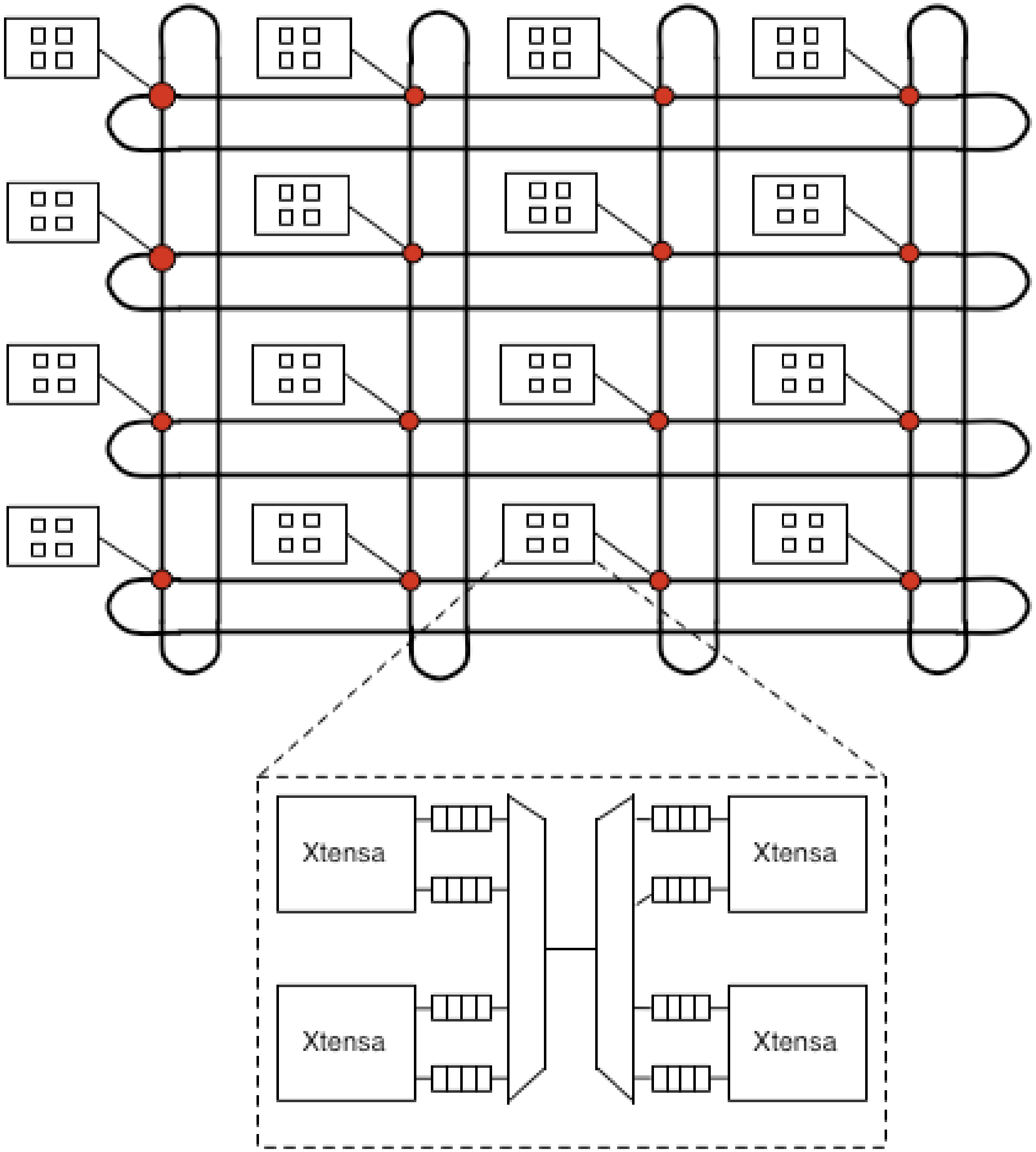
Architectural Support for Pmodels
Make hardware easier to program!



NVRAM (FLASH) for fault resilience

- **Logical topology is a full crossbar**
- **Each local store mapped to global address space**
- **To initiate a DMA transfer between processors:**
  - **Processors exchange starting addresses through TIE Queue interface**
    - **Optimized for small transfers**
  - **When ready, copy done directly from LS to LS**
  - **Copy will bypass cache hierarchy**

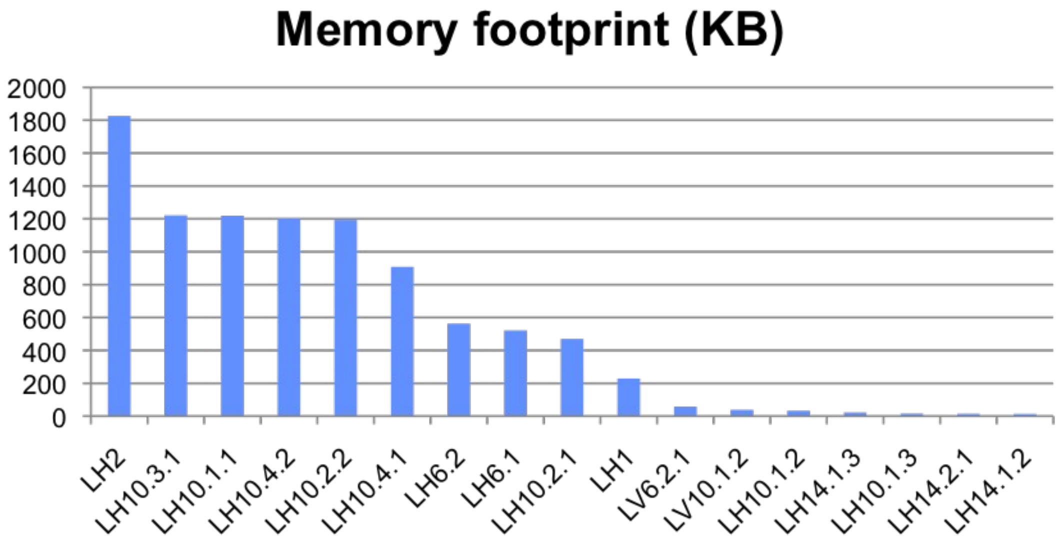
CMP Architecture - Physical View

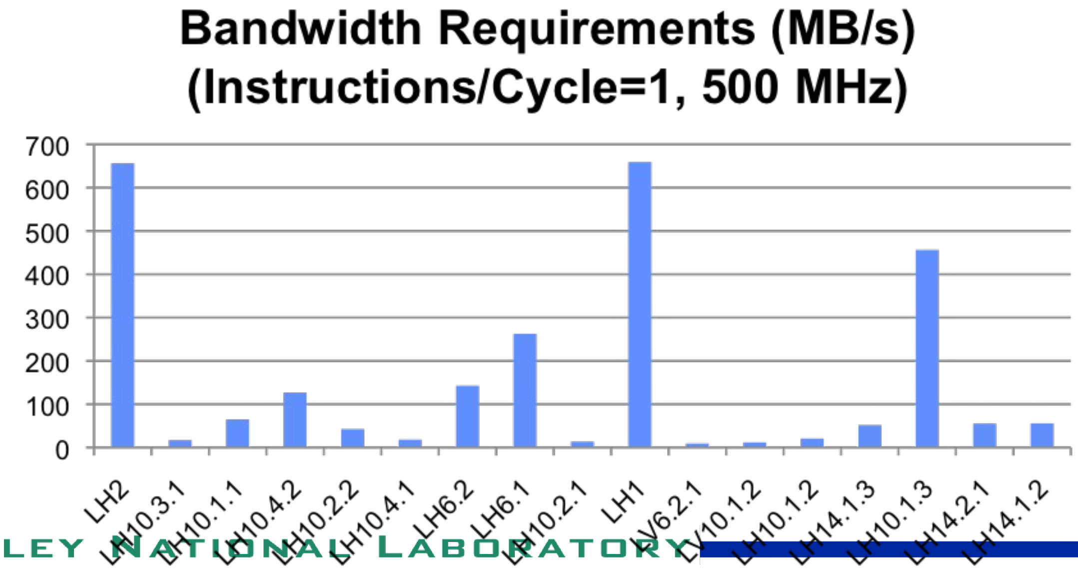


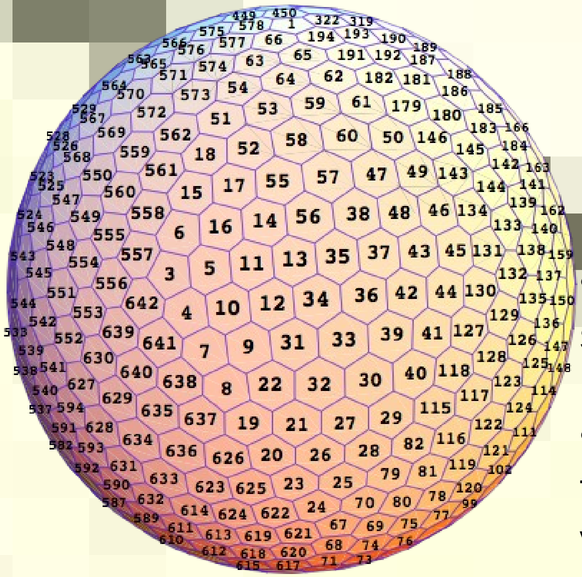
- Concentrated torus
  - Direct connect between 4 processors on a tile
  - Packet switched network connecting tiles
- Between 64 and 128 processors per die

Memory: Perhaps we *don't* need 1 Byte/FLOP *(Scripted Memory Movement)*

- Trace analysis key to memory requirements
  - Actually running the code gives realistic values for memory footprint, temporal reuse, DRAM bandwidth requirements
- Memory footprint: unique addresses accessed ➔ size of local store needed
- Temporal reuse: maximum number of addresses which will be reused at any time ➔ size of cache needed
- DRAM bandwidth
  - (instruction throughput) X (memory footprint)/ (instruction count)







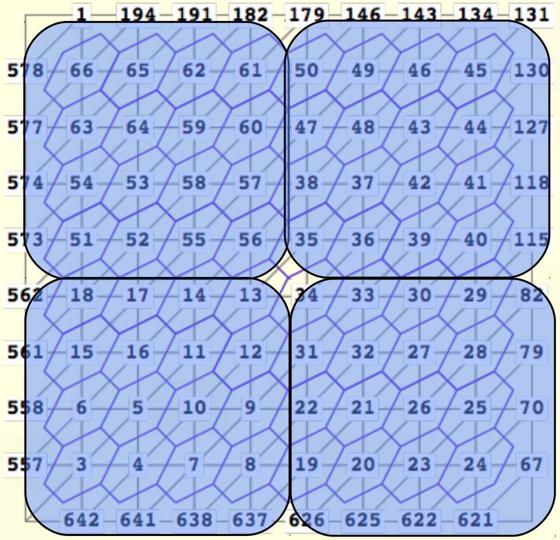
Discretization

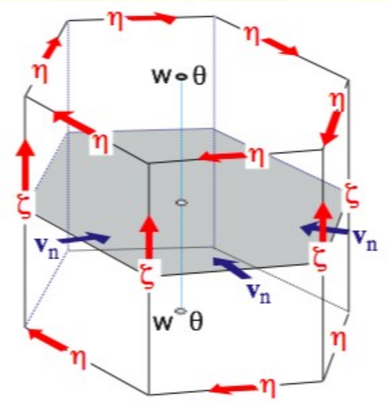
128 vertical levels

20M horizontal

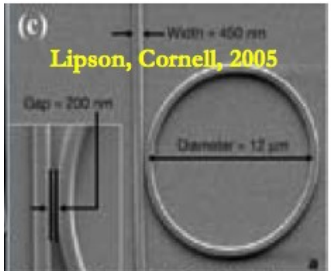
Design Trade-offs

- pack fewer cores in socket to minimize memory bandwidth
- maximize cores in socket to minimize surface-to-volume ratio

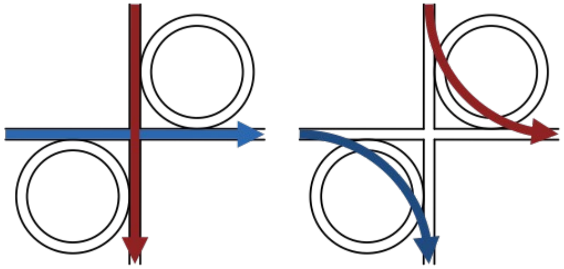




Silicon Photonics for Energy-Efficient Communication

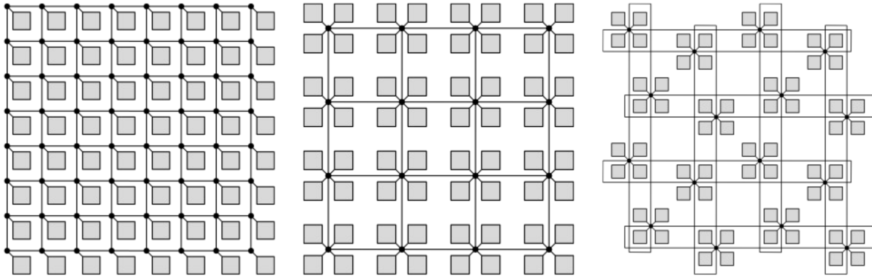


Silicon Photonic Ring Resonator



(a) Off state

(b) On state

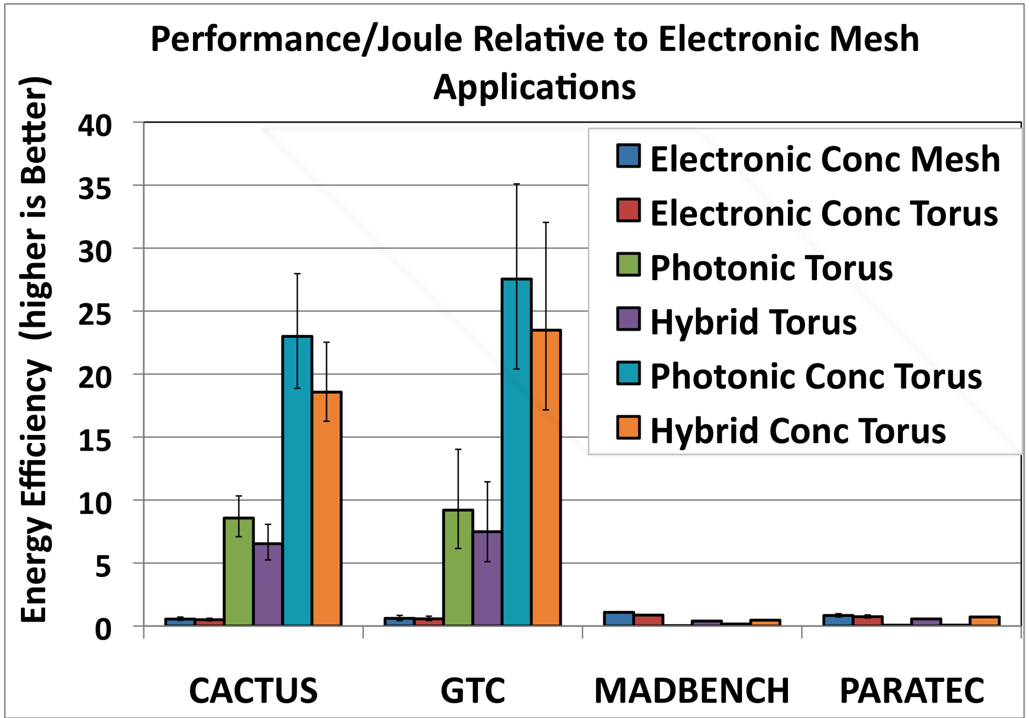


(a) Mesh

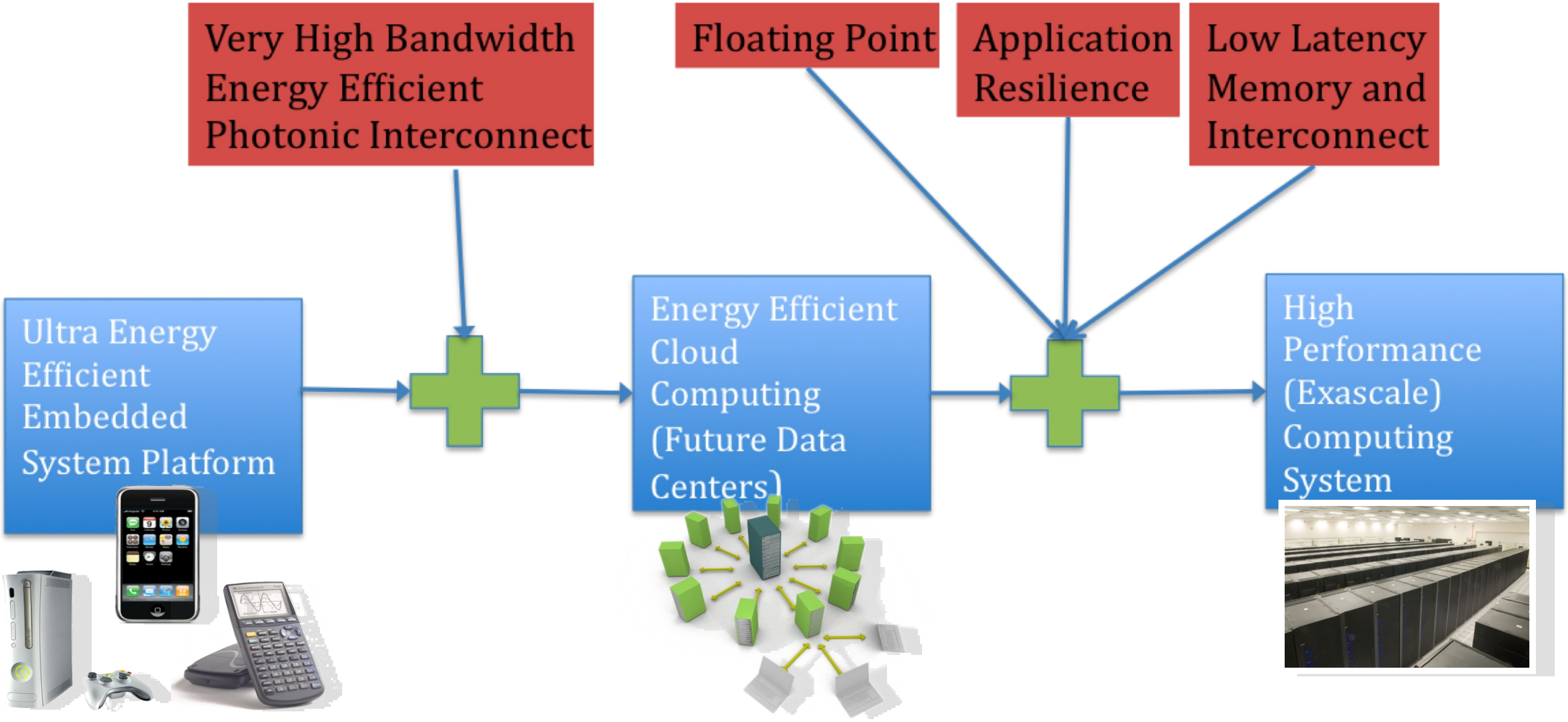
(b) Concentrated Mesh

(c) Concentrated Torus

- **Silicon photonics enables optics to be integrated with conventional CMOS**
- **Enables up to 27x improvement in communication energy efficiency!**



Technology Continuity for A Sustainable Hardware Ecosystem



Need building blocks for a compelling environment at all scales

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, scientific algorithms together using hardware emulation and auto-tuning
  - This is the right way to design efficient HPC systems!

Apply approach to broad range of Exascale-class scientific applications

Our Approach

- **Identify target applications FIRST**
  - –Demonstrate using Climate Application (**Green Flash**)

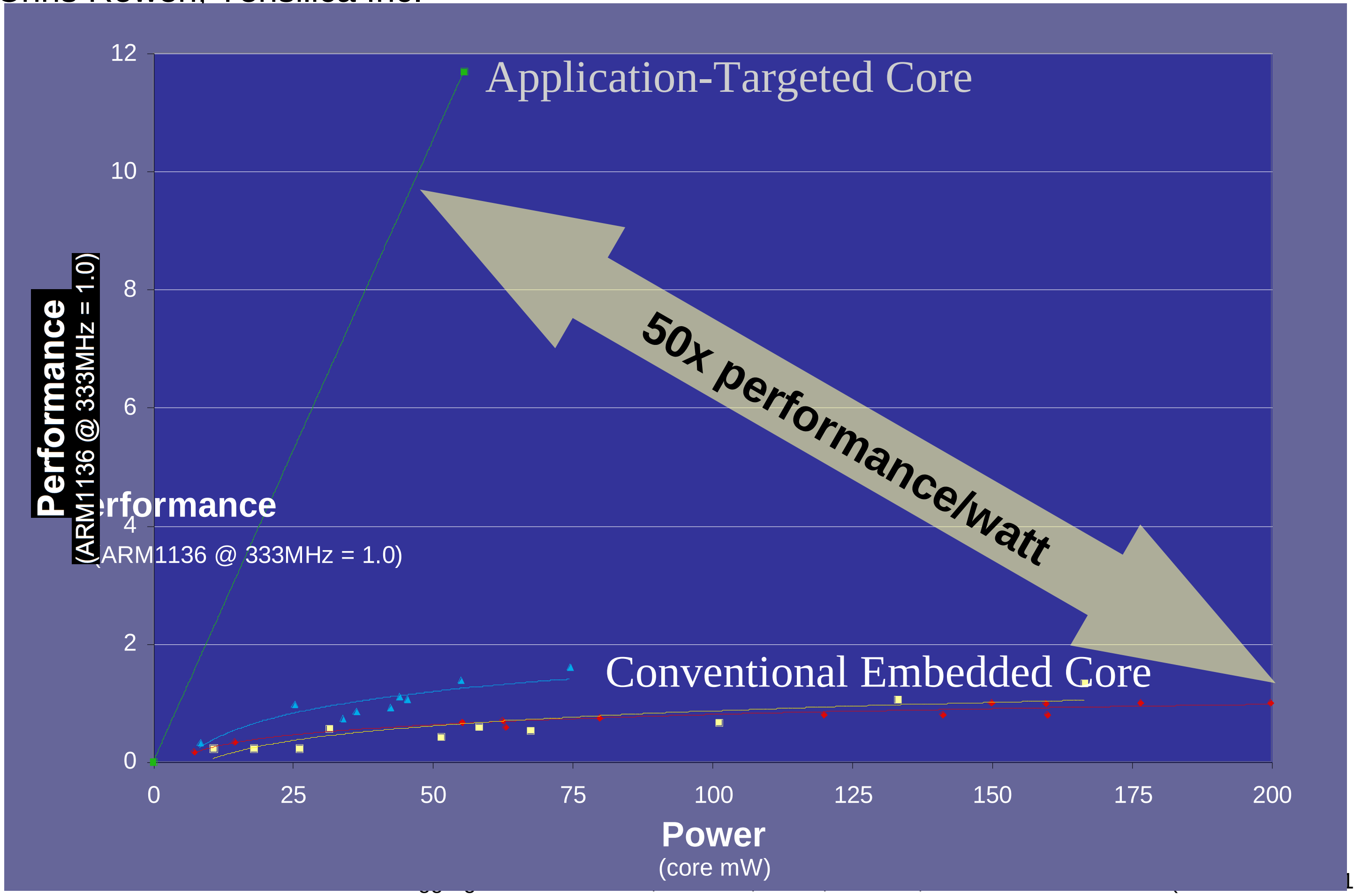
- **Tailor system to requirements of target scientific problem**
  - –Use design principles from embedded computing

- **Tightly couple hardware/software/science development**
  - –Simulate hardware before you build it (RAMP)
  - –Use applications as the test, not kernels (V&V)
  - –Automate software tuning process (AutoTuning)

Processor Power and Performance

Embedded Application-Specific Cores

Courtesy of Chris Rowen, Tensilica Inc.



1), ARM1026EJ-S, Tensilica Diamond 570T, T1050 and T1030, MIPS 20K, NECVR5000). MIPS M4K, MIPS 4Ke, MIPS 4Ks, MIPS 24K, ARM 968E-S, ARM 966E-S, ARM926EJ-S, ARM7TDMI-S scaled by ratio of Dhrystone MIPS within architecture family. All power figures from vendor websites, 2/23/2006.