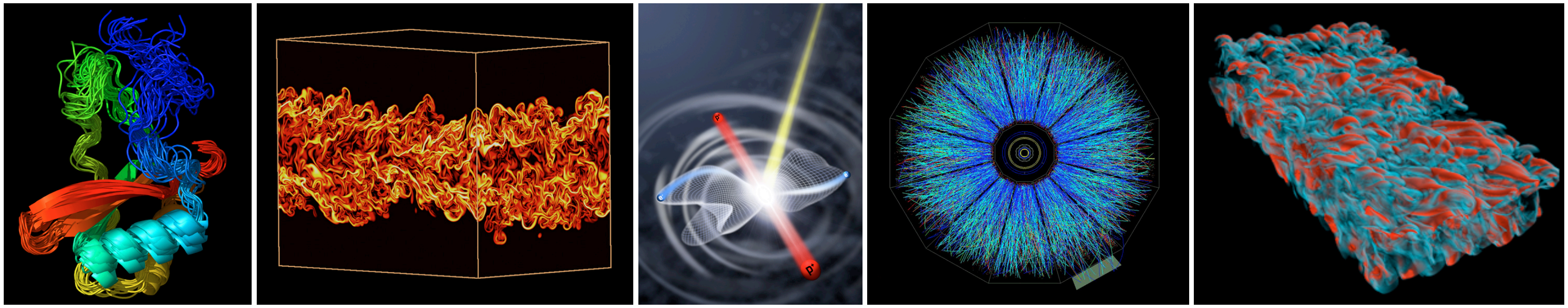


Green Flash and Co-Design: Energy Efficient Exascale Systems

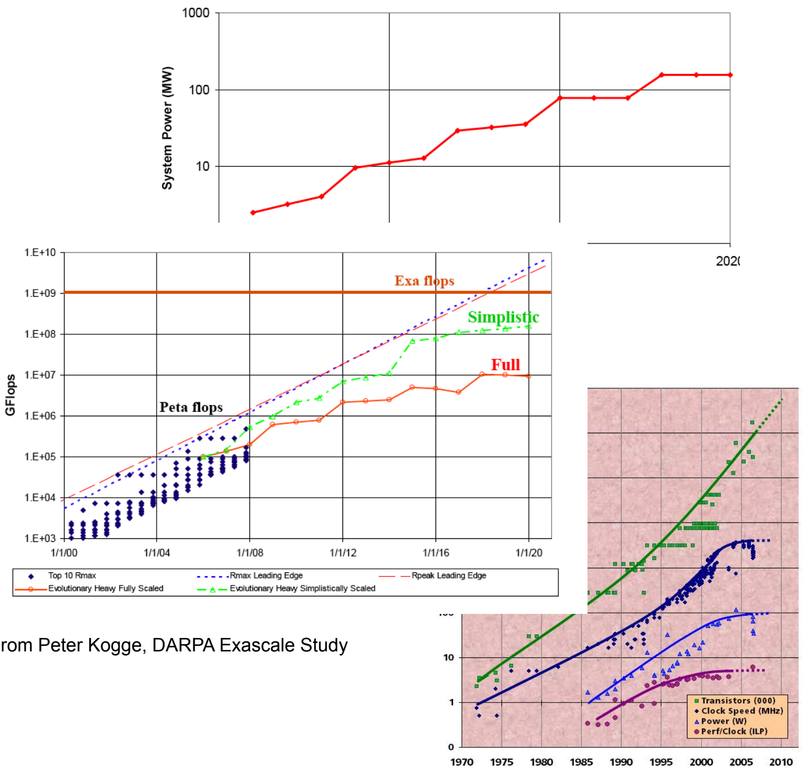
HPC and Cloud Computing Workshop

June 16, 2011

David Donofrio

Motivations… A Quick Review

- Power primary design constraint
- Frequency Wall
- Memory Wall
- Programming models
- No clear path forward



From Peter Kogge, DARPA Exascale Study

Olukotun et. al.

The Challenge

- ***Power is the leading design constraint in HPC system design***

- ***How to get build an exascale system without building a nuclear power plant next to my HPC center?***

- ***How can you assure the systems will be balanced for a reasonable science workload?***

- ***How do you make it "programmable?"***

Technology Investment Trends

- 1990s - R&D computing hardware dominated by desktop/COTS
  - **Had to learn how to use COTS technology for HPC**

- 2010 - R&D investments moving rapidly to consumer electronics/ embedded processing
  - **Must learn how to leverage embedded/consumer processor technology for future HPC systems**

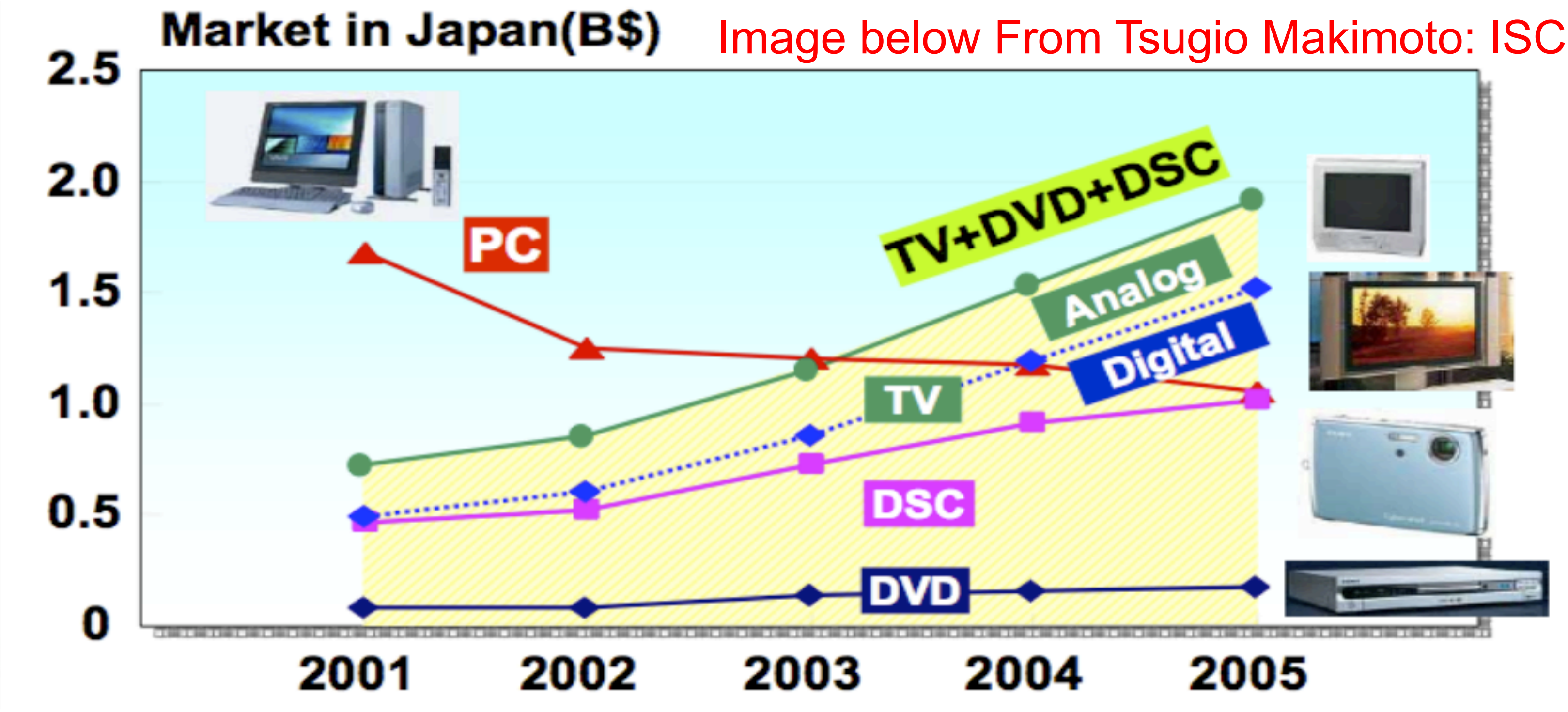
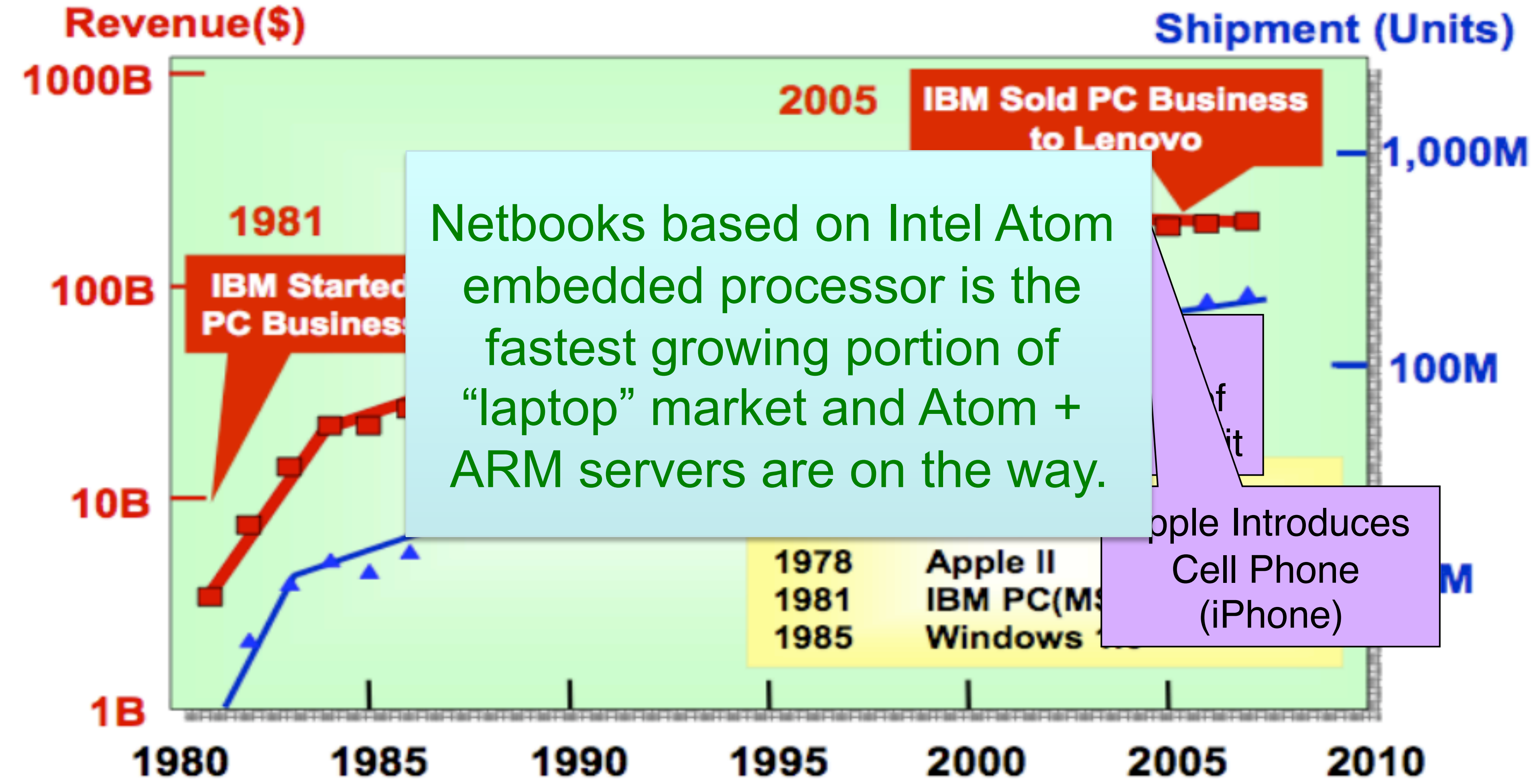


Image below From Tsugio Makimoto: ISC2006

Consumer Electronics has Replaced PCs as the Dominant Market Force in CPU Design



Netbooks based on Intel Atom embedded processor is the fastest growing portion of “laptop” market and Atom + ARM servers are on the way.

pple Introduces Cell Phone (iPhone)

Source: IDC

Embedded / HPC Synergy

- High Performance embedded is aligned with HPC
  - HPC used to be performance without regard to power
  - Now HPC is power limited (max delivered performance/watt)
  - Embedded has always been driven by max performance/watt (max battery life) and minimizing cost ($1 cell phones)
  - Now HPC and embedded requirements are aligned
- Your “smart phone” is driving technology development
  - Desktops are no longer in the drivers seat
  - This is not a bad thing because high-performance embedded has longer track record of application-driven design
  - Hardware/Software co-design comes from embedded design

Redefining “commodity”

- Must use “commodity” technology to build cost-effective design
- The primary cost of a chip is development of the intellectual property
  - Mask and fab typically 10% of NRE in embedded
  - Design and verification dominate costs
  - Embedded computing has a vibrant market for IP/ circuit-design (pre-verified, place & route)
  - Redefine your notion of “commodity”!

The ‘chip’ is not the commodity…

The stuff you put on the chip is the commodity

CoDesign to Reduce Waste

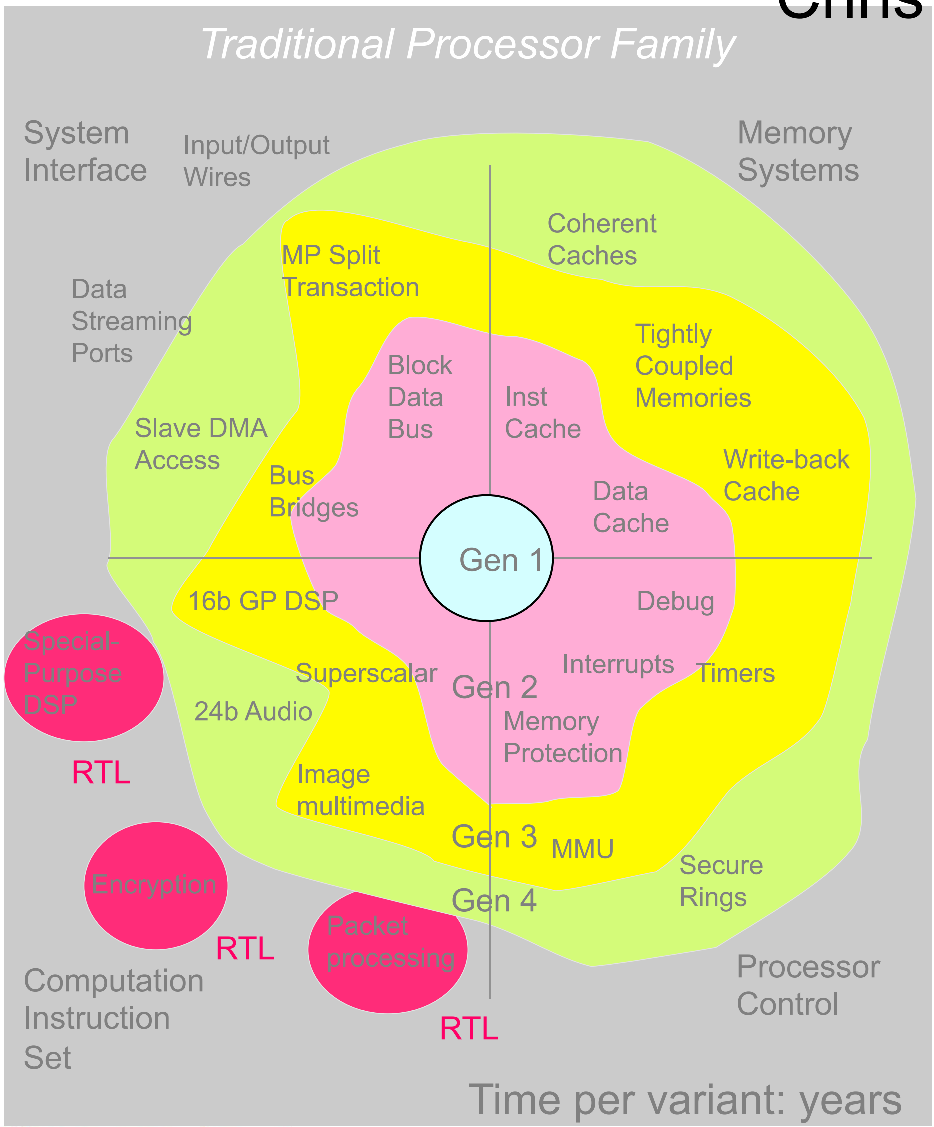
Biggest win was in what we do NOT include in an HPC Design

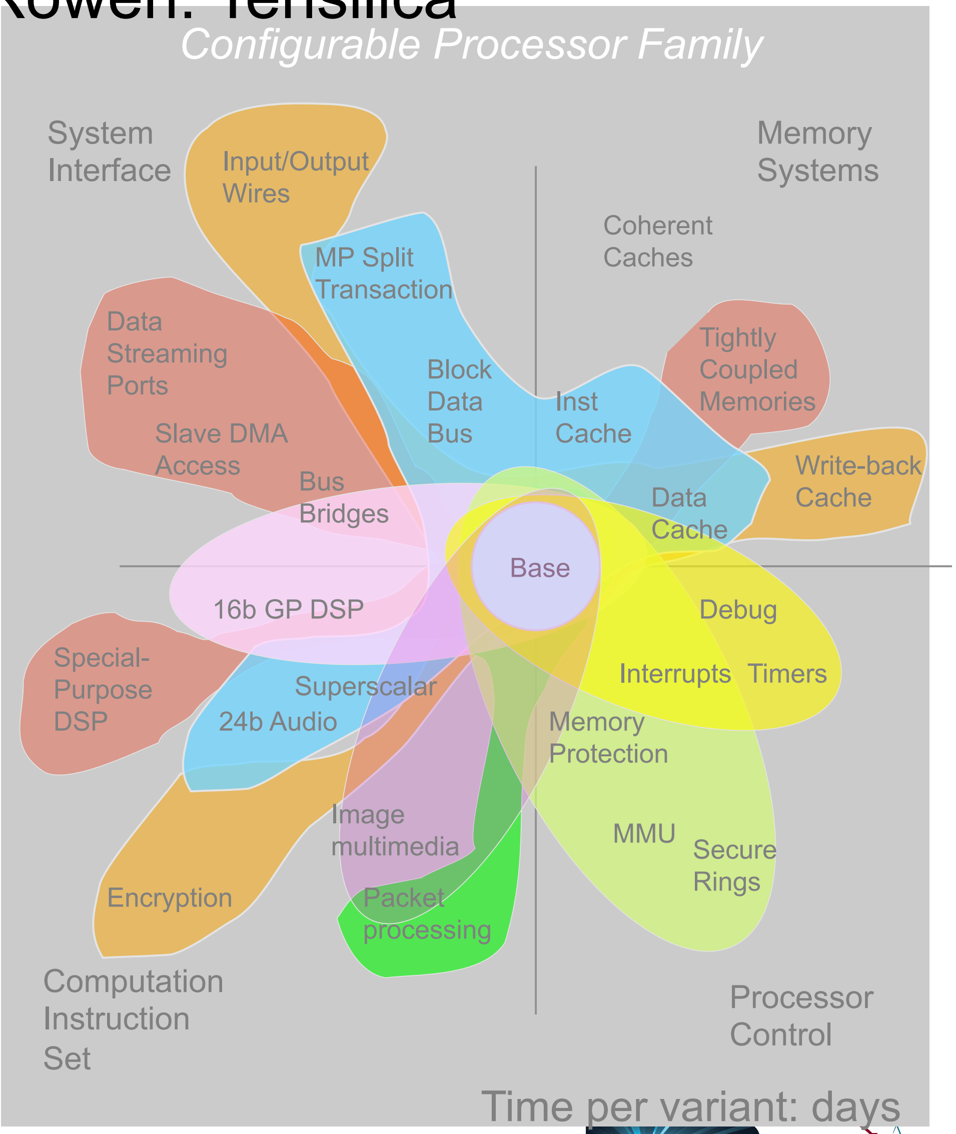
Mark Horowitz 2007*: "Years of research in low-power embedded computing have shown only one design technique to reduce power: reduce waste."*

Seymour Cray 1977: *"Don't put anything in to a supercomputer that isn't necessary."*

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

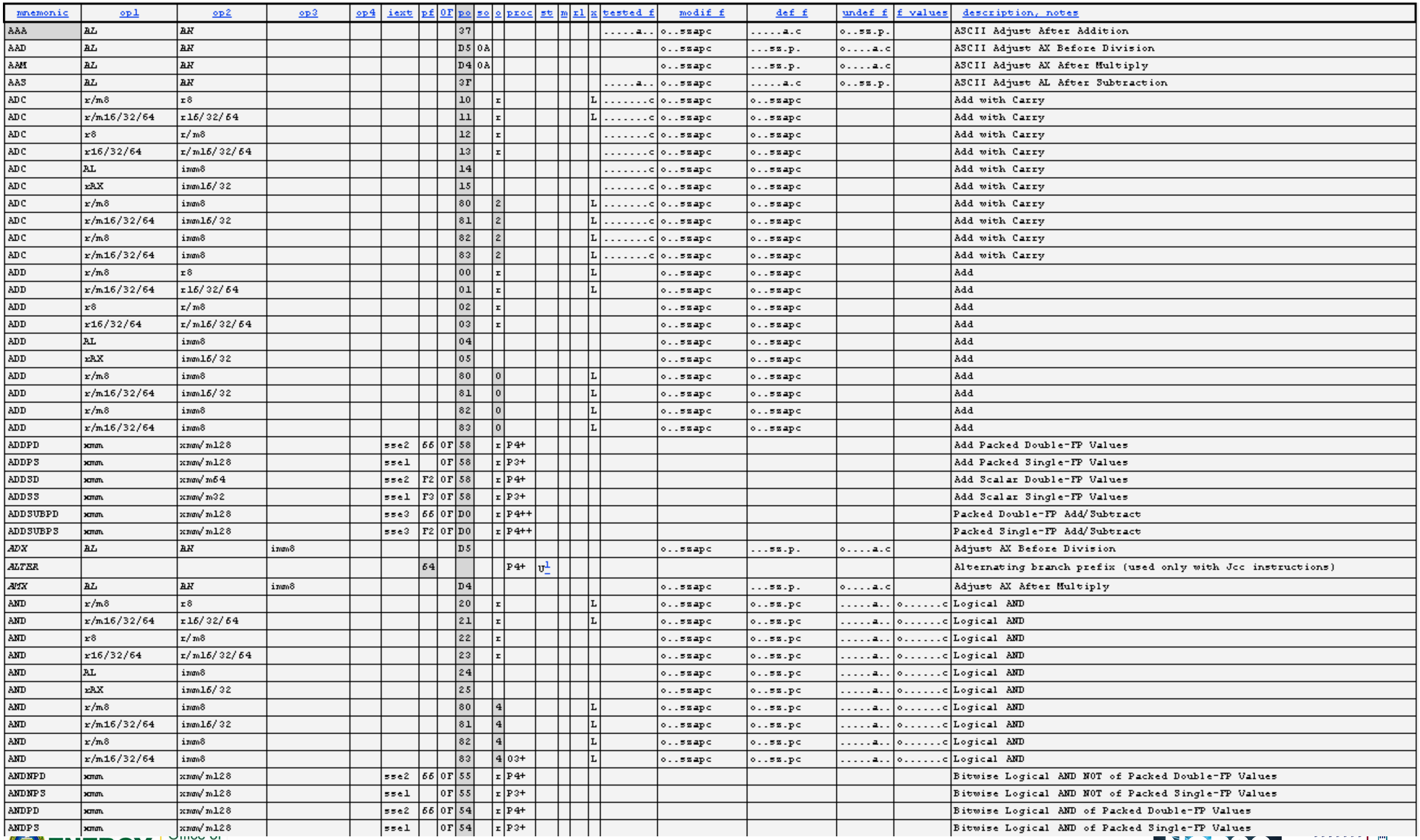
Chris Rowen: Tensilica





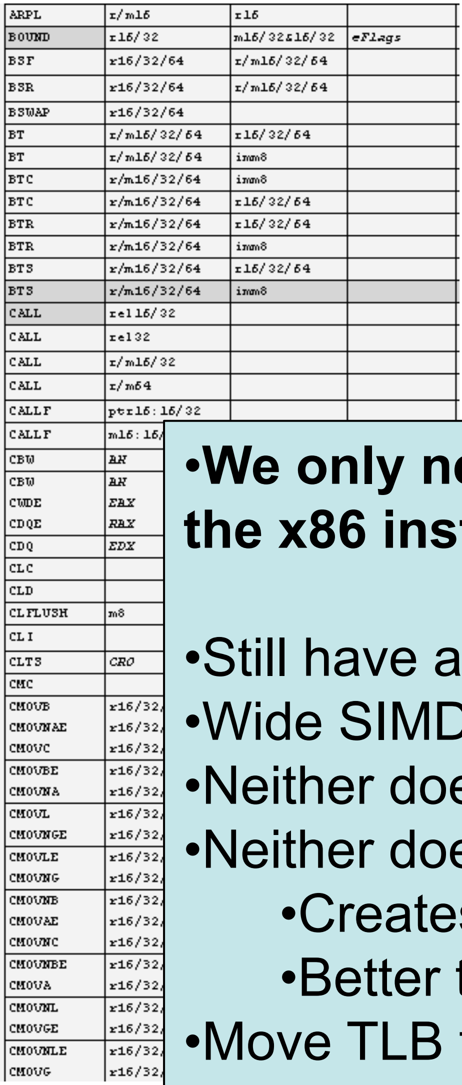
⬭ Area = silicon cost and power

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



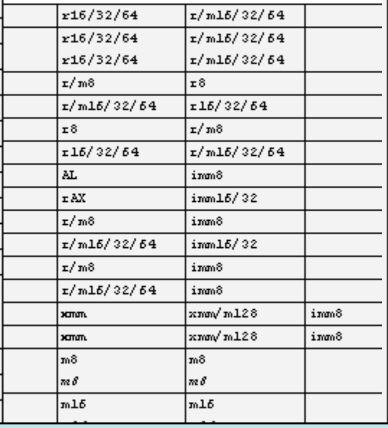
mnemonic	op1	op2	op3	op4	iext	pf	0F	po	so	o	proc	st	m	rl	x	tested f	modif f	def f	undef f	f values	description, notes
AAA	AL	AH						37								.....a..	o..szapc	.....a.c	o..sz.p.		ASCII Adjust After Addition
AAD	AL	AH						D5	0A								o..szapc	...sz.p.	o....a.c		ASCII Adjust AX Before Division
AAM	AL	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 Unused Opcodes

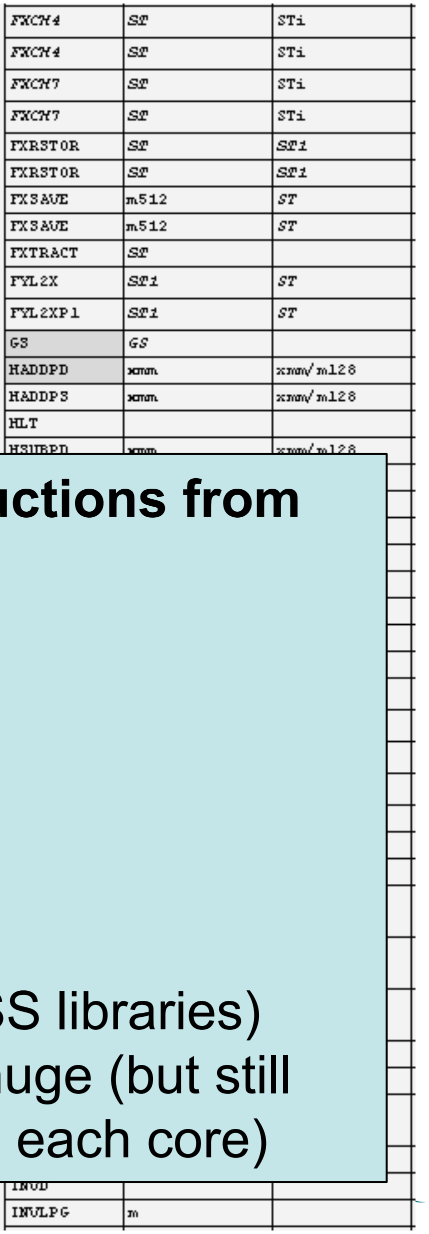


ARPL	r/m16	r16	
BOUND	r16/32	m16/32&16/32	eFlags
BSF	r16/32/64	r/m16/32/64	
BSR	r16/32/64	r/m16/32/64	
BSWAP	r16/32/64		
BT	r/m16/32/64	r16/32/64	
BT	r/m16/32/64	imm8	
BTC	r/m16/32/64	imm8	
BTC	r/m16/32/64	r16/32/64	
BTR	r/m16/32/64	r16/32/64	
BTR	r/m16/32/64	imm8	
BTS	r/m16/32/64	r16/32/64	
BTS	r/m16/32/64	imm8	
CALL	rel16/32		
CALL	rel32		
CALL	r/m16/32		
CALL	r/m64		
CALLF	ptr16:16/32		
CALLF	m16:16/		
CBW	AH		
CBW	AX		
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	CVTPS2PD	xmm	xmm/m128	
CMO	CVTPS2PI	mm	xmm/m64	
CMO	CVTSD2SI	r32/64	xmm/m64	
CMP	CVTSD2SS	xmm	xmm/m64	
CMP	CVTSI2SD	xmm	r/m32/64	
CMP	CVTSI2SS	xmm	r/m32/64	
CMP	CVTSS2SD	xmm	xmm/m32	
CMP	CVTSS2SI	r32/64	xmm/m32	
CMP	CVTTPD2DQ	xmm	xmm/m128	
CMP	CVTTPD2PI	mm	xmm/m128	
CMP	CVTTPS2DQ	xmm	xmm/m128	
CMP	CVTTPS2PI	mm	xmm/m64	
CMP	CVTTSD2SI	r32/64	xmm/m64	
CMP	CVTTSS2SI	r32/64	xmm/m32	
CMP	CWD	DX	AX	
CMP	CWD	DX	AX	
CMP	CDQ	EDX	EAX	
CMP	CQO	RDX	RAX	
	CWDE	EAX	AX	
	DAA	AL		
	DAS	AL		



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



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	
INVD			
INVLPG	m		

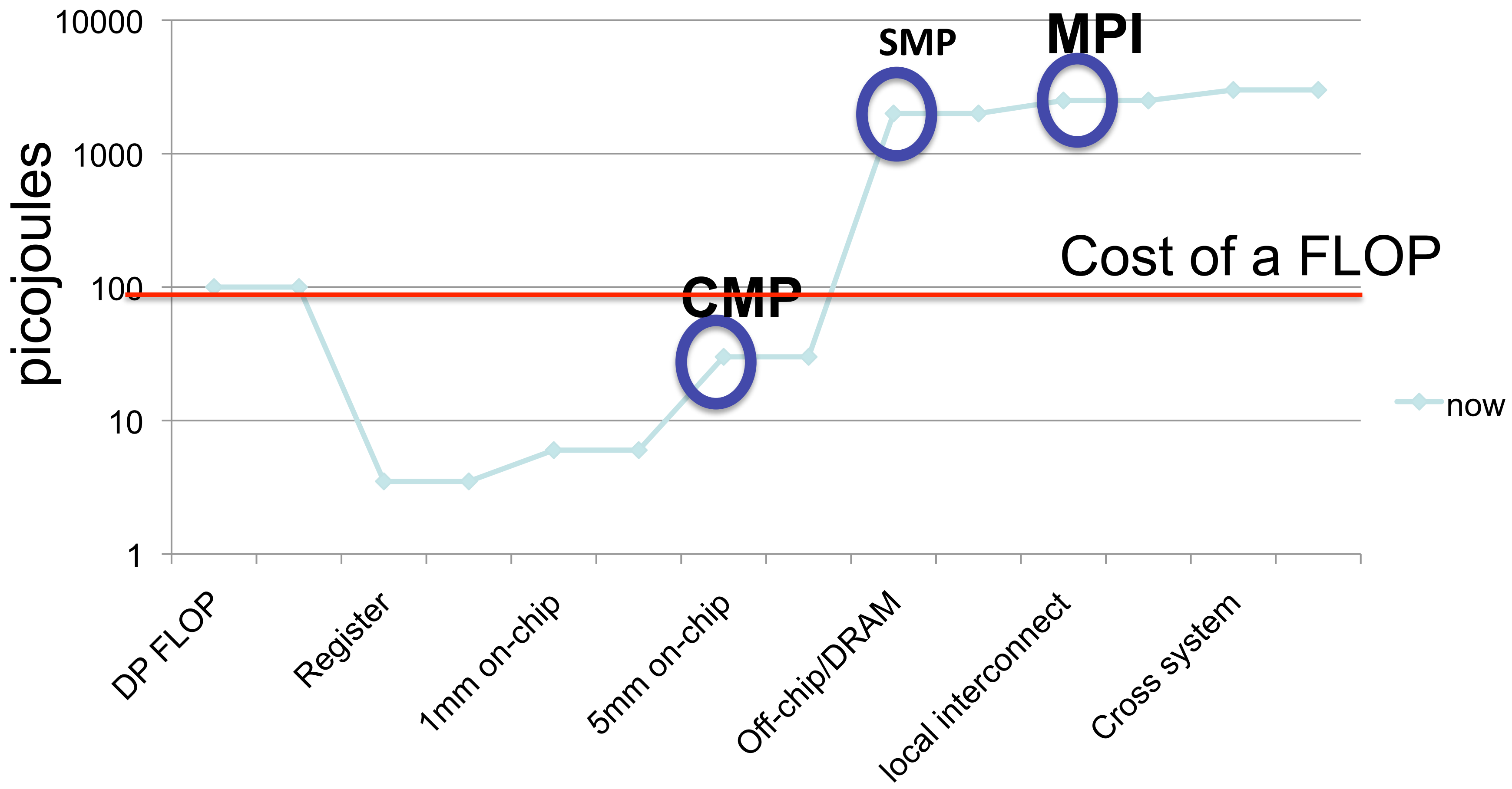
- **We only need 80 out of the nearly 300 ASM instructions from the x86 instruction set!**

- 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)

The Cost of Data Movement Data Locality Management and

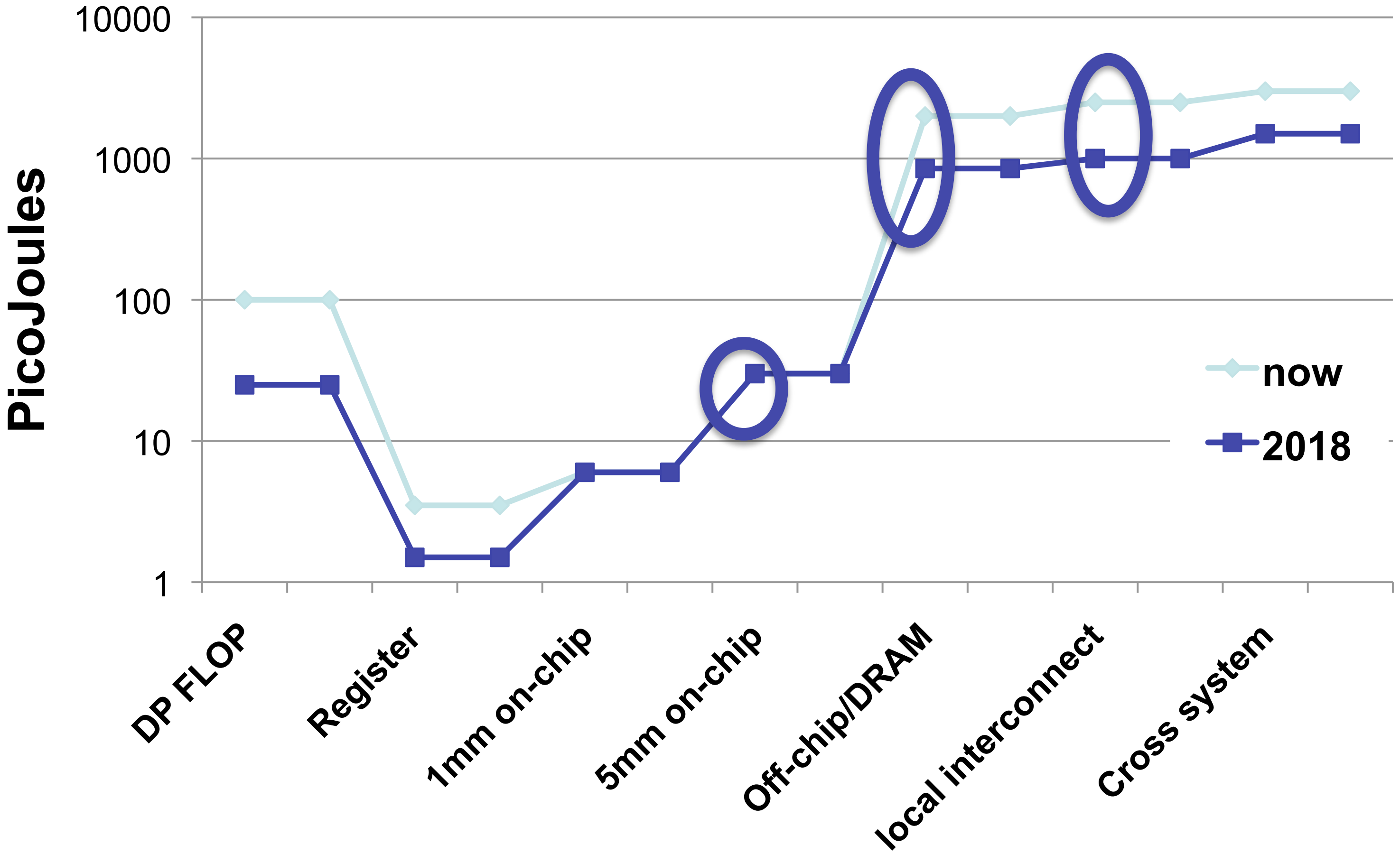
How do those cores talk to each other?

The Cost of Data Movement



The Situation Will Not Improve in 2018

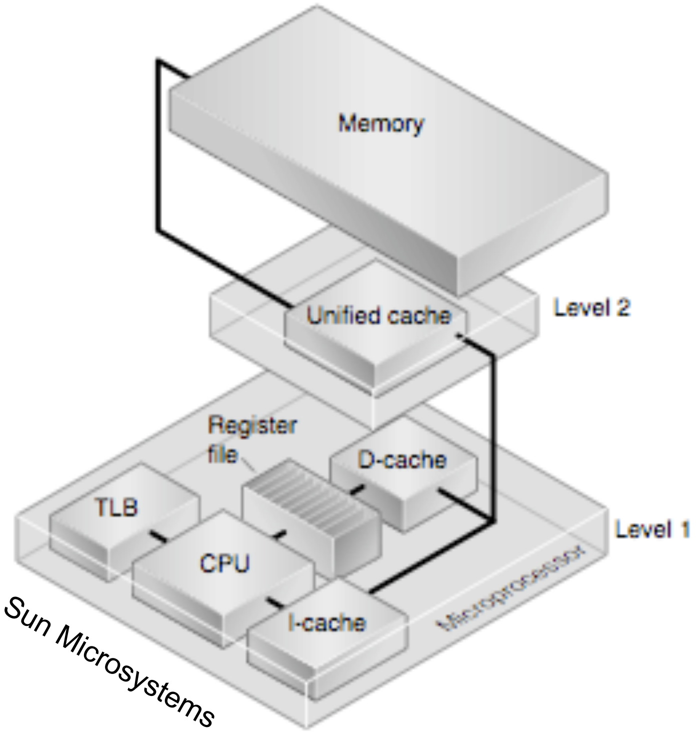
Energy Efficiency will require careful management of data locality



Important to know when you are on-chip and when data is off-chip!

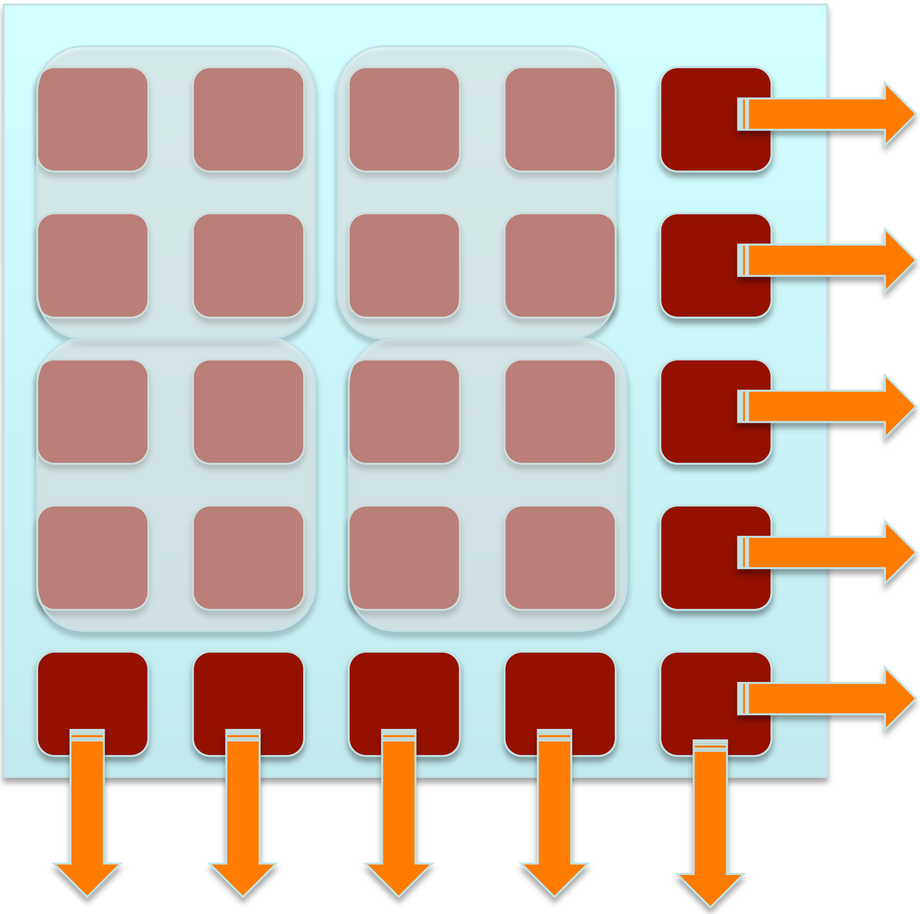
Data Locality Management

Vertical Locality Management



Sun Microsystems

Horizontal Locality Management



Data Locality Management

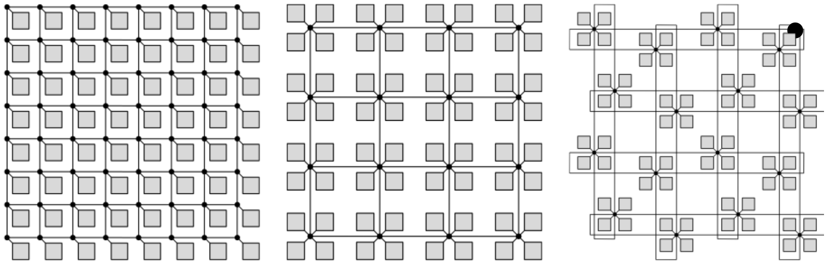
Vertical Locality Management

- Movement of data up and down cache hierarchy
  - Cache virtualizes notion of on-chip off-chip
  - Software managed memory (local store) is hard to program (cell)
- Virtual Local Store / Malleable Memory
  - Use conventional cache for portability
  - Only use SW managed memory only for performance critical code
  - Repartition as needed

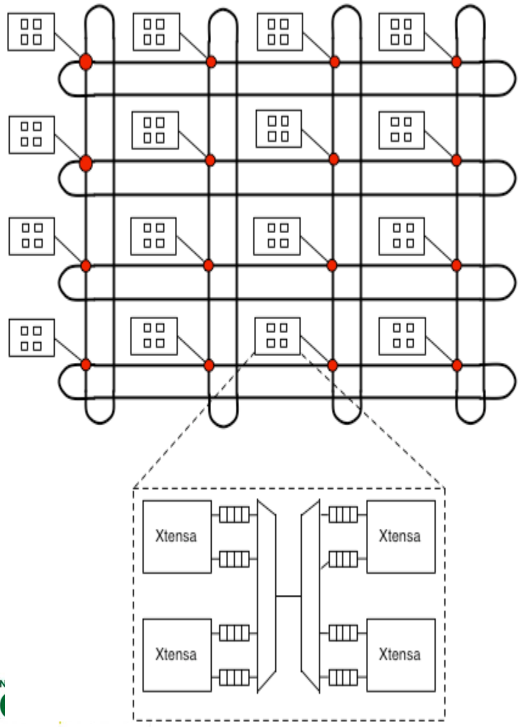
Horizontal Locality Management

- Movement of data between processors
  - 10x lower latency and 10x higher bandwidth on-chip
  - Need to minimize distance of horizontal data movement
- Encode Horizontal locality into memory address
  - Hardware hierarchy where high-order bits encode cabinet and low-order bits encode chip-level distance
- Map local-store into global address space
  - Hierarchical Partitioned Global Address space

Network-on-Chip (NoC) Architecture

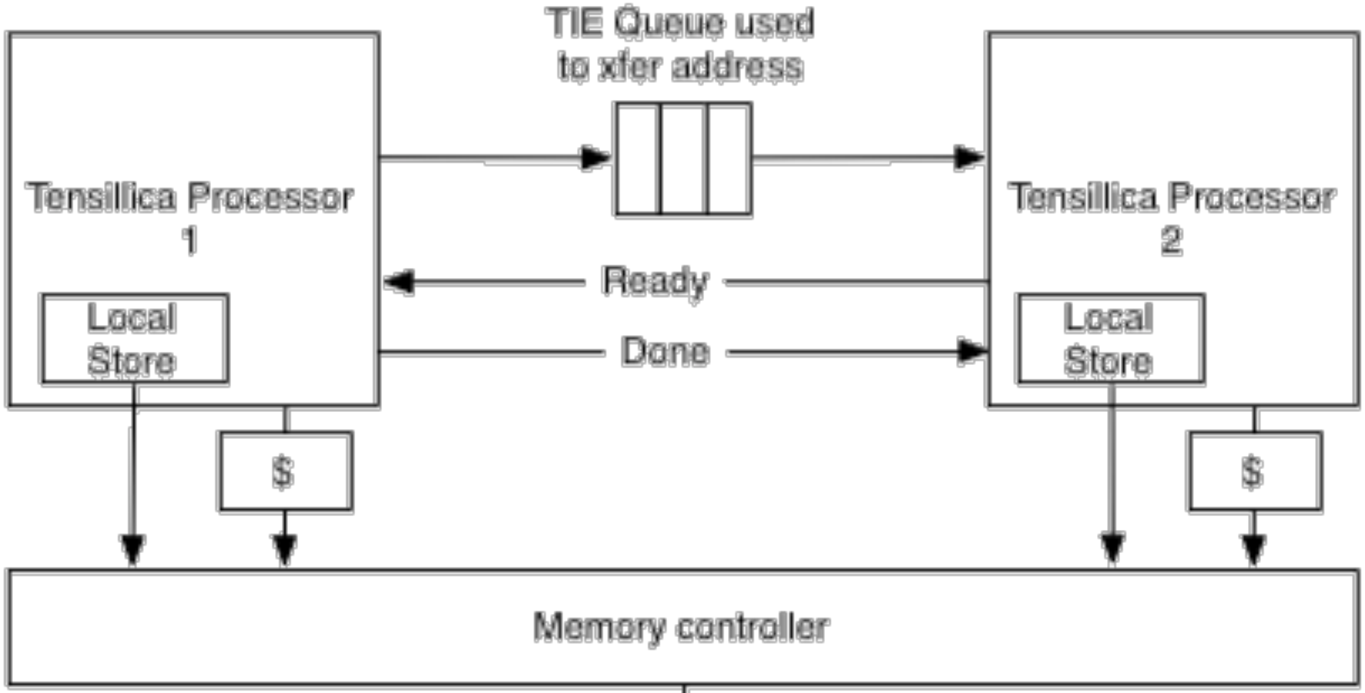


(a) Mesh (b) Concentrated Mesh (c) Concentrated Torus



- Fabric to connect cores and "options" together on chip
  - IPC for cores
  - Connects to Memory controllers
  - Connects to optional devices
  - Can choose NoC topology
- Science optimized communication mechanisms
  - All core-local memories are mapped into memory address space (PGAS on a chip)
  - Direct message queues between cores
  - Fine-grained sync for memory consistency

Providing Direct Support for Data Locality Control



- Each Processor Tile has a conventional Cache + Local Store
  - *Enables incremental porting to local store*
- Has direct inter-processor message queues (trivial with Tie Queues)
  - Enables direct inter processor communication for low-overhead synchronization
  - Can be used for very efficient DAG Scheduling

This is just utilizing off-the-shelf technology from embedded space!

Green Flash: Overview

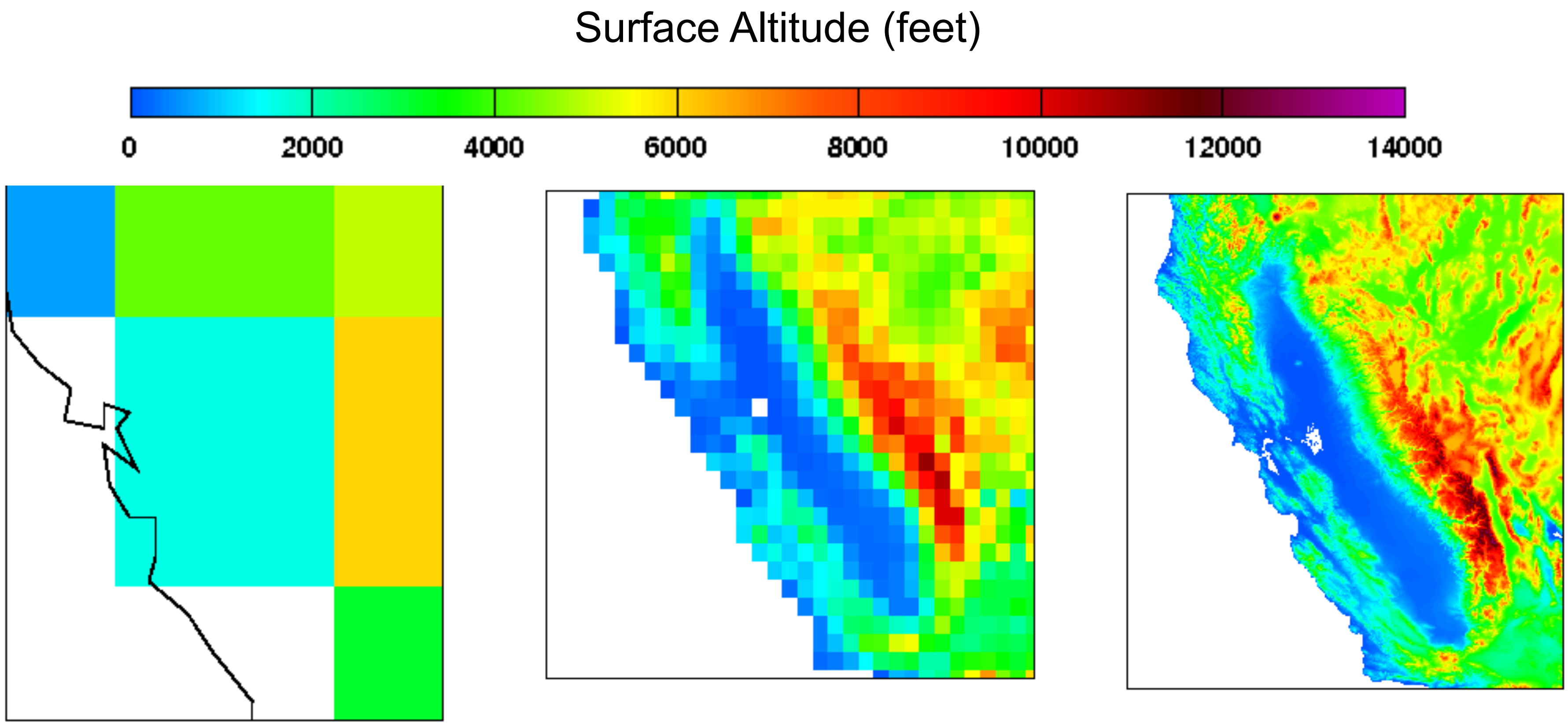
John Shalf, PI

- We present an alternative approach to developing systems to serve the needs of scientific computing
  - Choose our **science target first** to drive design decisions
  - Leverage new technologies driven by consumer market
  - ***Auto-tune software*** for performance, productivity, and portability
  - Use hardware-accelerated architectural emulation to rapidly prototype designs (***auto-tune the hardware too!***)
- Requires a holistic approach: Must innovate algorithm/ software/hardware together (Co-design)

Achieve 100x energy efficiency improvement over mainstream HPC approach

An Application Driver:
Global Cloud Resolving Climate Model

GCRM Status



200km
Typical resolution of
IPCC AR4 models

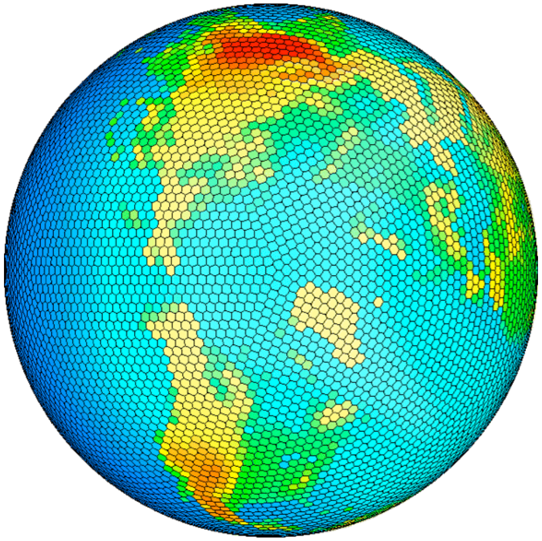
25km
Upper limit of climate models
with cloud parameterizations

1km
Cloud system resolving models
are a transformational change

Computational Requirements

- ~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*

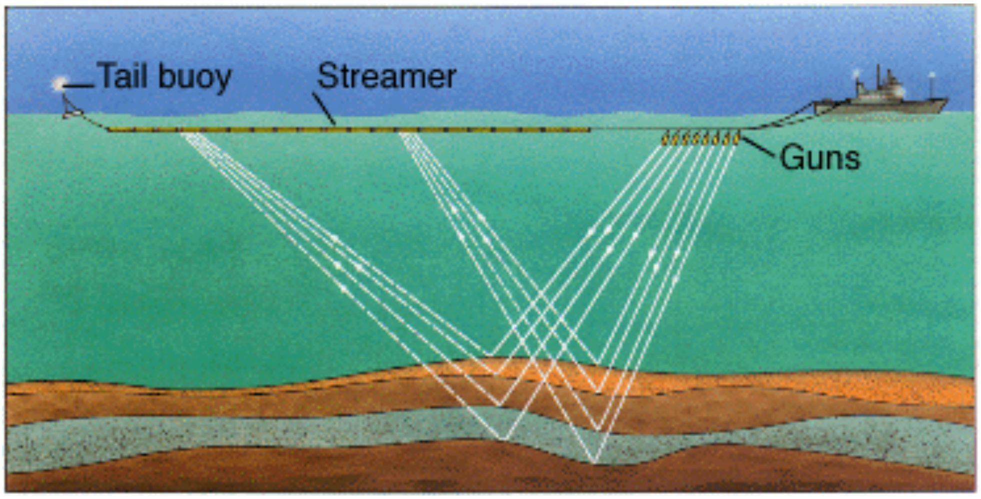
  CSU Icosahedral Code

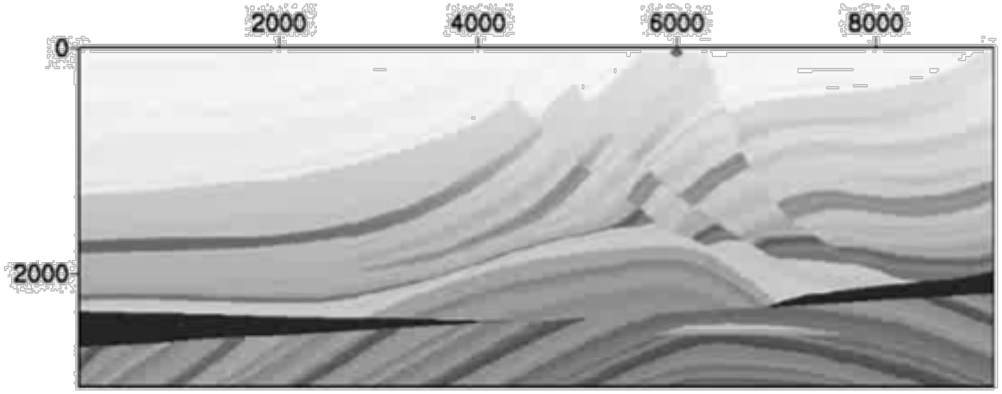


Must maintain 1000x faster than real time for practical climate simulation

An Application Driver:
Seismic Exploration

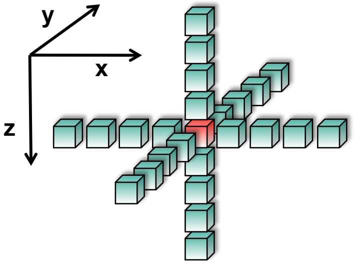
Example Design Study
Green Wave: Seismic Imaging





- Seismic imaging used extensively by oil and gas industry
  - Dominant method is RTM (Reverse Time Migration)
- RTM models acoustic wave propagation through rock strata using explicit PDE solve for elastic equation in 3D
  - High order (8th or more) stencils
  - High computational intensity

- **Typical survey requires months of computing on petascale-sized resources**



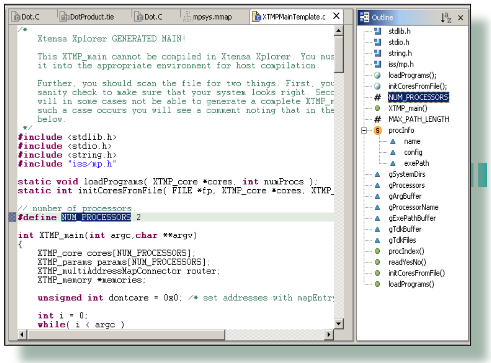
Embracing Embedded Tech

- Have most of the IP and experience with for low-power technology
- Have sophisticated tools for rapid turn-around of designs
- Vibrant commodity market in IP components
  - *Change your notion of "commodity"!*
  - *it's commodity IP on the chip (not the chip itself!)*

- Convergence with HPC requirements
  - **Need better computational efficiency and lower power**
  - **Now we both must face parallelism**

Embedded Design Automation

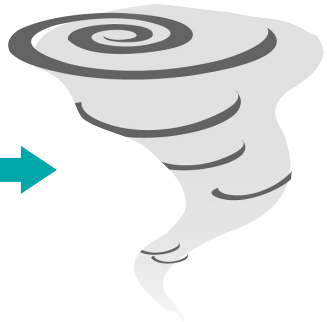
(Example from Existing Tensilica Design Flow)



```
/*
    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 )
```

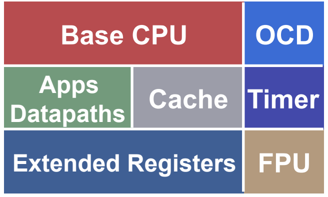
Processor configuration

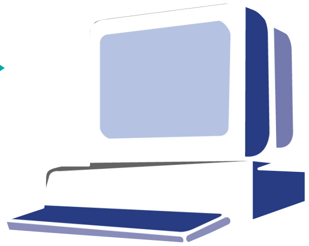
1. Select from menu
2. Automatic instruction discovery (XPRES Compiler)
3. Explicit instruction description (TIE)



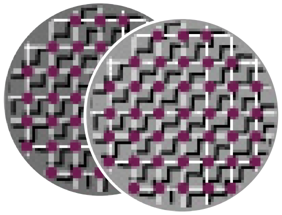
Processor Generator (*Tensilica*)

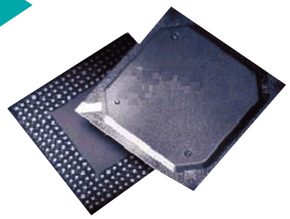
Application-optimized processor implementation (RTL/Verilog)





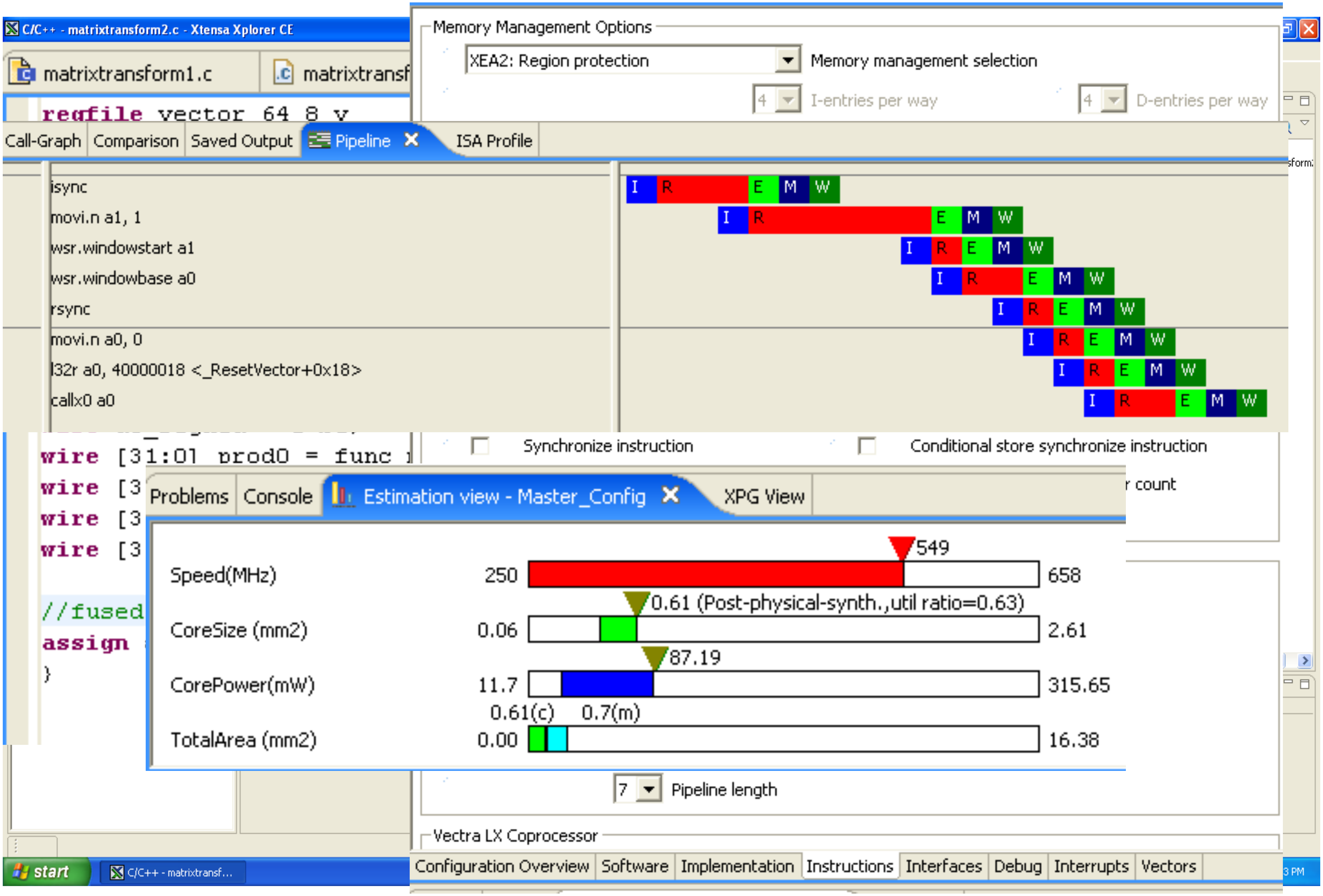
Tailored SW Tools: Compiler, debugger, simulators, Linux, other OS Ports *(Automatically generated together with the Core)*





Build with any process in any fab

Processor Generator



C/C++ - matrixtransform2.c - Xtensa Xplorer CE

matrixtransform1.c | matrixtransf

```
regfile vector 64 8 v
wire [31:0] prod0 = func 1
wire [3
wire [3
wire [3
//fused
assign
}
```

Memory Management Options

XEA2: Region protection — Memory management selection

4 I-entries per way   4 D-entries per way

Call-Graph | Comparison | Saved Output | Pipeline | ISA Profile

```
isync
movi.n a1, 1
wsr.windowstart a1
wsr.windowbase a0
rsync
movi.n a0, 0
l32r a0, 40000018 <_ResetVector+0x18>
callx0 a0
```

□ Synchronize instruction   □ Conditional store synchronize instruction

r count

Problems | Console | Estimation view - Master_Config | XPG View

	min	value	max
Speed(MHz)	250	549	658
CoreSize (mm2)	0.06	0.61 (Post-physical-synth.,util ratio=0.63)	2.61
CorePower(mW)	11.7	87.19	315.65
TotalArea (mm2)	0.00	0.61(c) 0.7(m)	16.38

7 Pipeline length

Vectra LX Coprocessor

Configuration Overview | Software | Implementation | Instructions | Interfaces | Debug | Interrupts | Vectors

Software Performance

Software Auto-tuning: Don't depend on a human to do a machine's job.

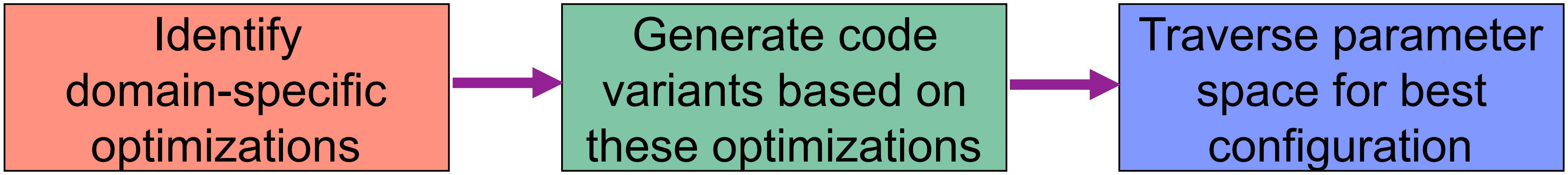
Auto-Tuning for Performance Portability

Challenge: How to optimize for multiple architectures

- Labor-intensive user optimizations for each specific architecture
- Different architectural solutions require vastly different optimizations
- Non-obvious interactions between optimizations & hardware

Solution: Auto-tuning

- Automate search across a complex optimization space
- Achieve performance far beyond current compilers
- Attain performance portability for diverse architectures



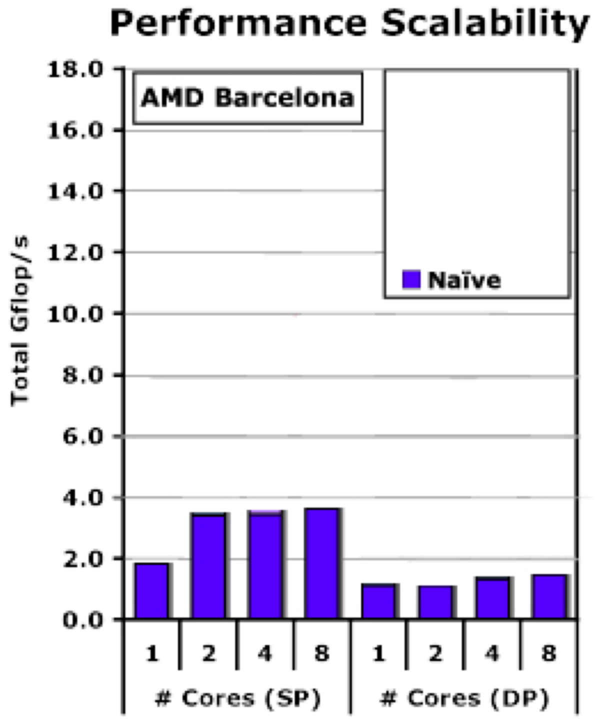
Identify domain-specific optimizations

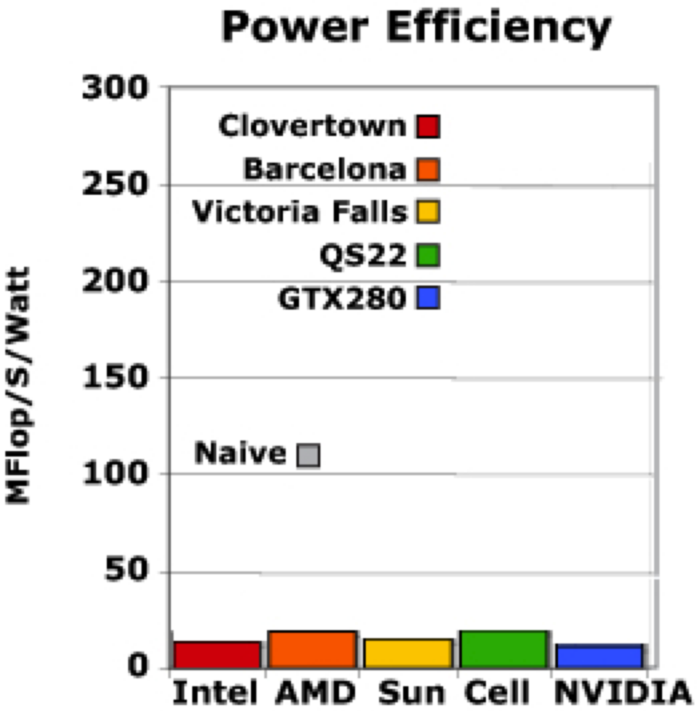
Generate code variants based on these optimizations

Traverse parameter space for best configuration

Auto-Tuning for Finite Difference

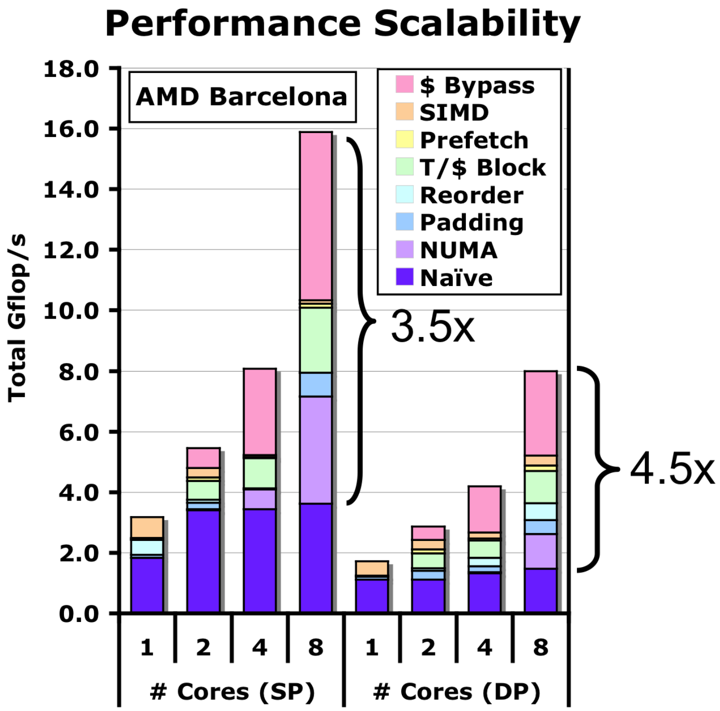
- Attains performance portability across different multicore designs
- Only requires basic compiling technology
- Achieve serial performance, scalability, optimized power efficiency

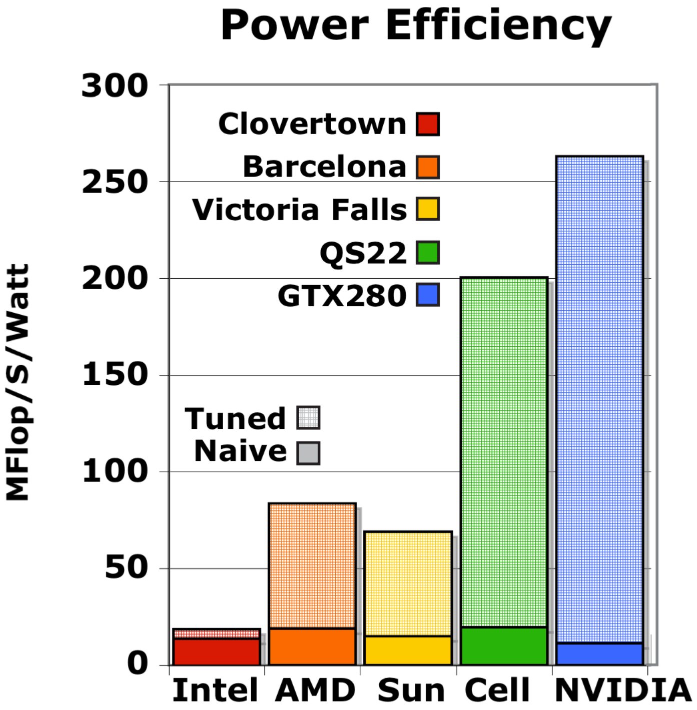




Auto-Tuning for Finite Difference

- Attains performance portability across different multicore designs
- Only requires basic compiling technology
- Achieve serial performance, scalability, optimized power efficiency



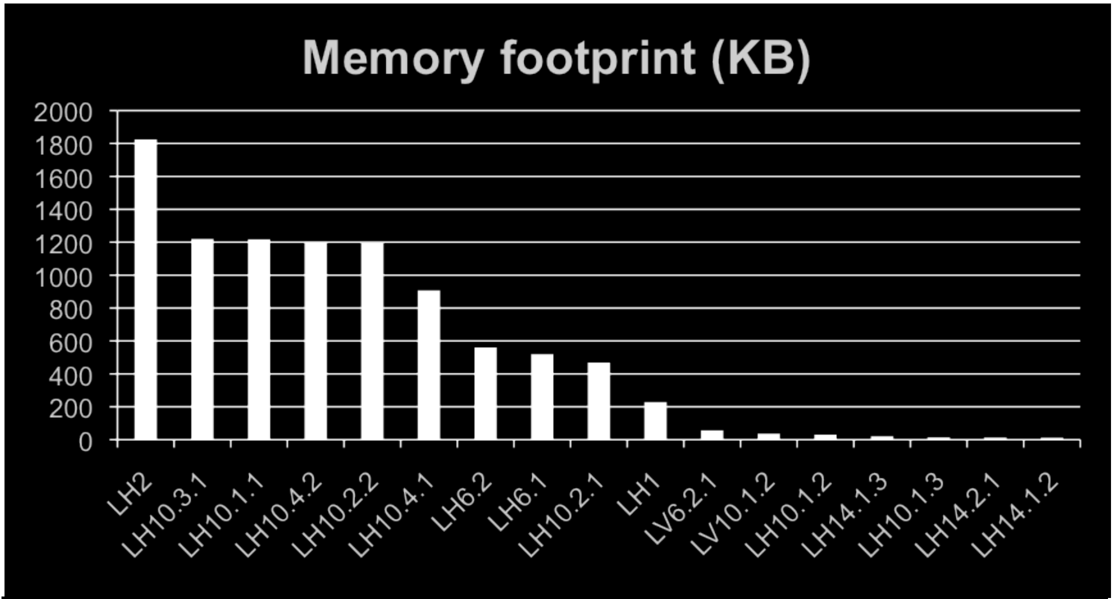


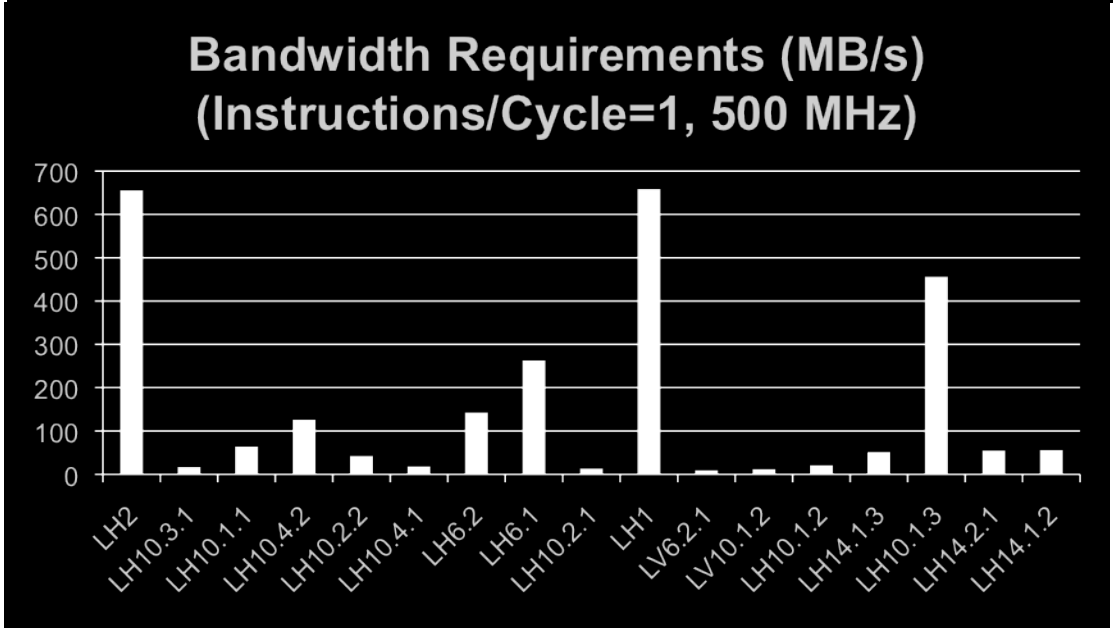
How well does this work?

Putting everything together…

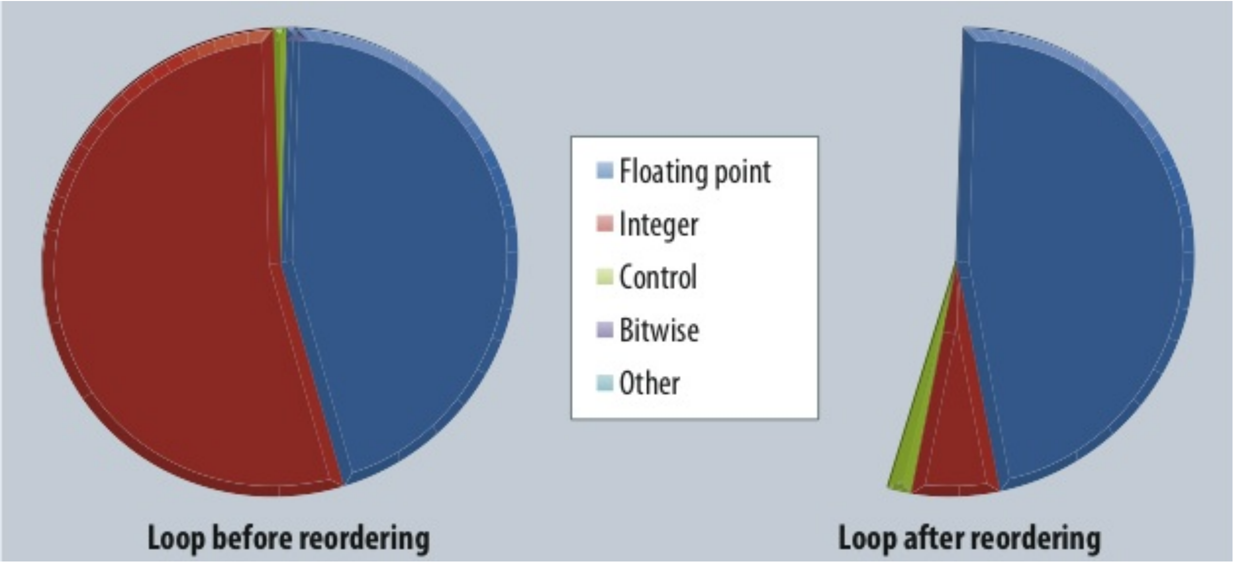
Example Design Study: Climate

- Analyze each loop within climate code
  - Extract temporal reuse and bandwidth requirements
- Use traces to determine cache size and DRAM BW requirements
- Ensure memory hierarchy can support application


Memory footprint (KB)


Bandwidth Requirements (MB/s)
(Instructions/Cycle=1, 500 MHz)

Example Design Study: Climate



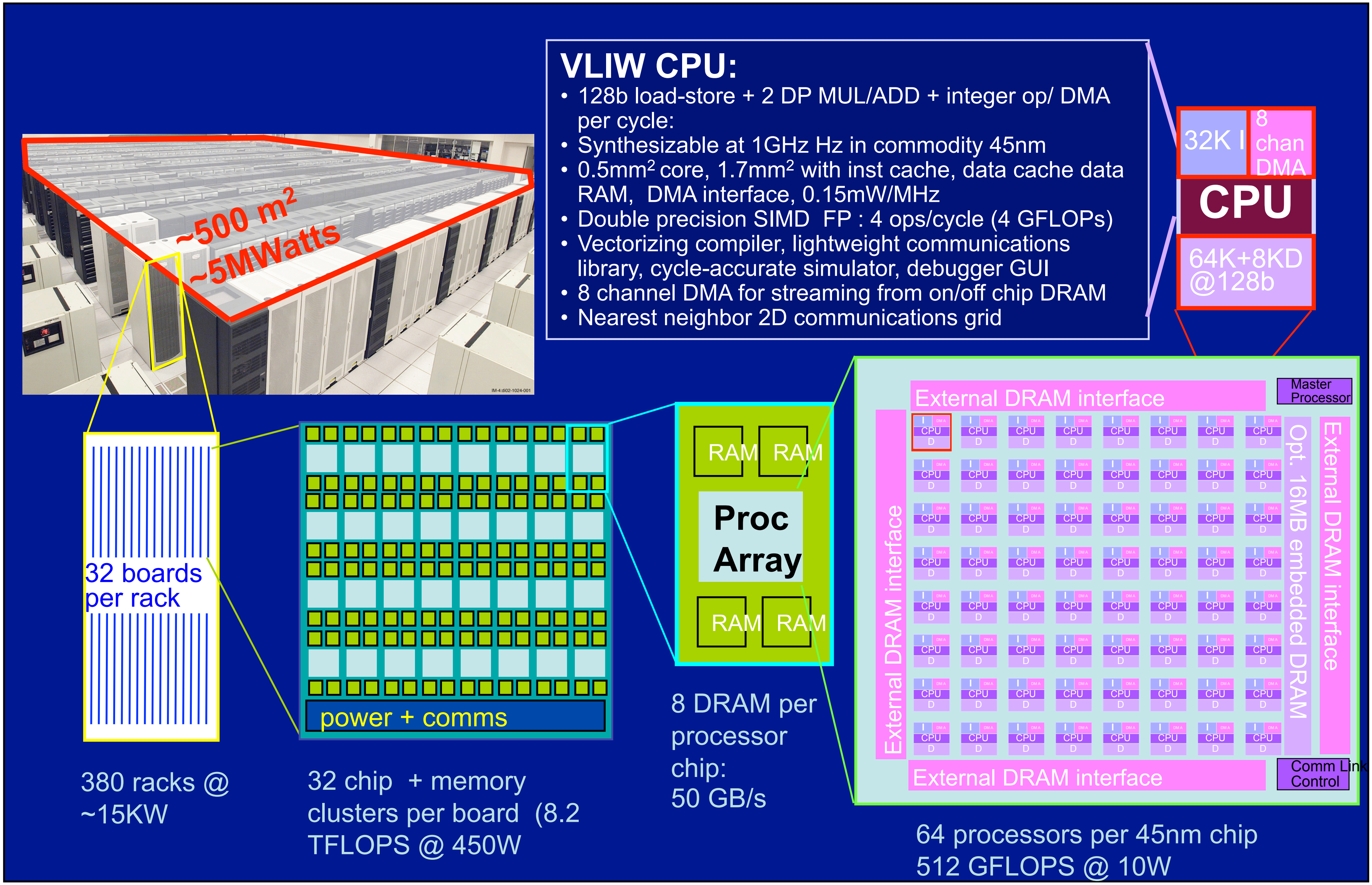
- Original code:
  - 160KB Cache requirement
  - < 50% FP Instructions

- Tuned Code:
  - 1KB Cache Requirement
  - > 85% FP instructions

Loop optimization resulted in 160x reduction in cache size and a 2x increase in execution speed

Climate Modeling System

Strawman 200PF Design



VLIW CPU:

- 128b load-store + 2 DP MUL/ADD + integer op/ DMA per cycle:
- Synthesizable at 1GHz Hz in commodity 45nm
- 0.5mm^2 core, 1.7mm^2 with inst cache, data cache data RAM, DMA interface, 0.15mW/MHz
- Double precision SIMD FP : 4 ops/cycle (4 GFLOPs)
- Vectorizing compiler, lightweight communications library, cycle-accurate simulator, debugger GUI
- 8 channel DMA for streaming from on/off chip DRAM
- Nearest neighbor 2D communications grid

380 racks @ ~15KW

32 chip + memory clusters per board (8.2 TFLOPS @ 450W

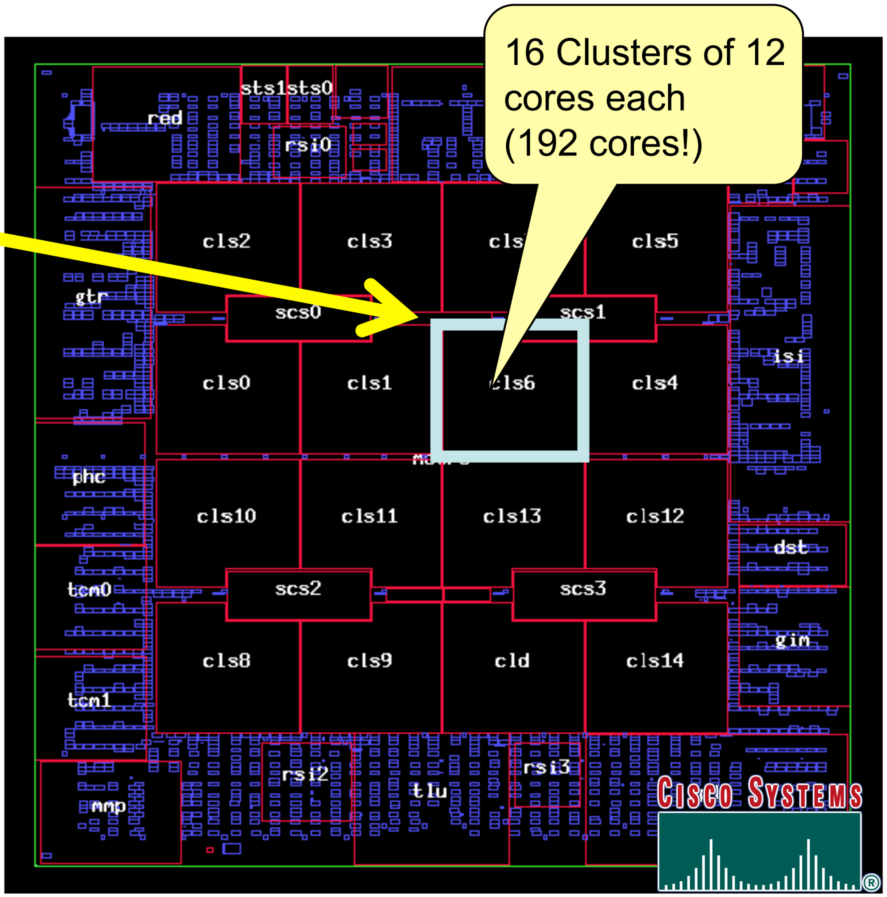
8 DRAM per processor chip: 50 GB/s

64 processors per 45nm chip 512 GFLOPS @ 10W

Green Flash: Fault Tolerance/Resilience

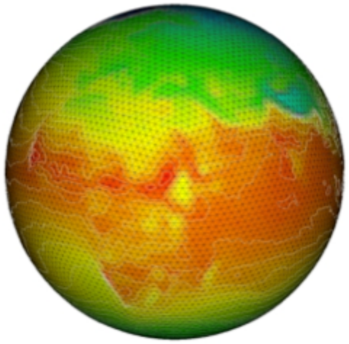
- *Large scale applications must tolerate node failures*
- Our design does not expose unique risks
  - Faults proportional to sockets (not cores) & silicon surface area
  - Low-power manycore uses less surface area and fewer sockets

- **Hard Errors**
  - Spare cores in design (Cisco Metro: 188 cores + 8 spares)
  - SystemOnChip design (fewer components→fewer sockets)
- **Soft Errors**
  - ECC for memory and caches
  - On-board NVRAM controller for localized checkpoint

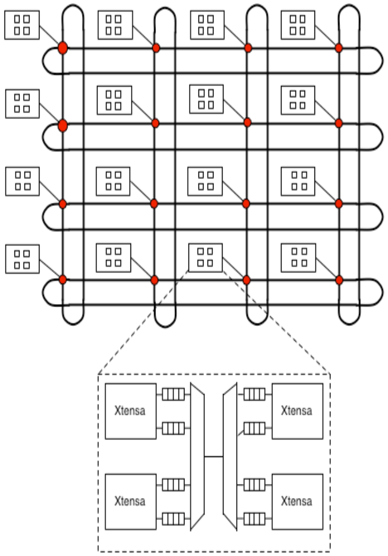

16 Clusters of 12 cores each (192 cores!)

Co-Design Before its Time

- Green Flash Demo'd during SC '09
- CSU atmospheric model ported to low-power core design
  - Dual Core Tensilica processors running atmospheric model at 25MHz
  - MPI Routines ported to custom Tensilica Interconnect
- Memory and processor Stats available for performance analysis
- Emulation performance advantage
  - 250x Speedup over merely function software simulator
- *Actual code running - not representative benchmark*

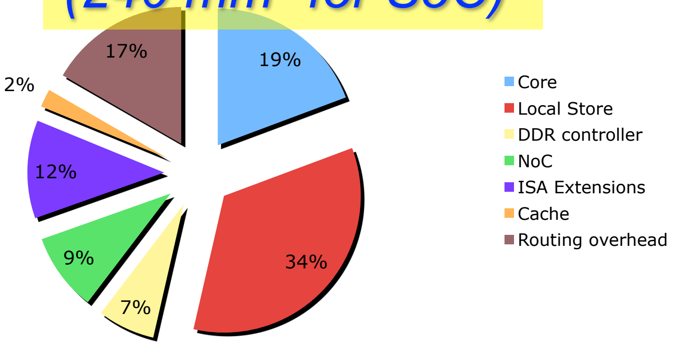


Icosahedral mesh for algorithm scaling

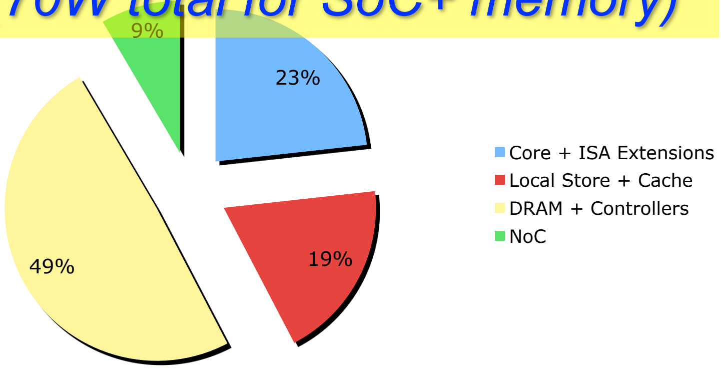


Example Design Study
Green Wave: Seismic Imaging

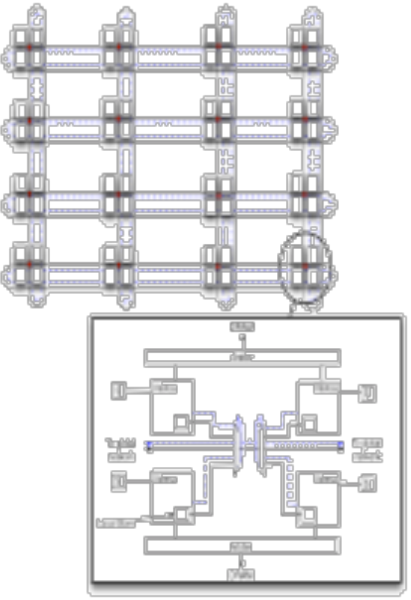
Area Breakdown
(240 mm^2 for SoC)



Power Breakdown
(70W total for SoC+ memory)

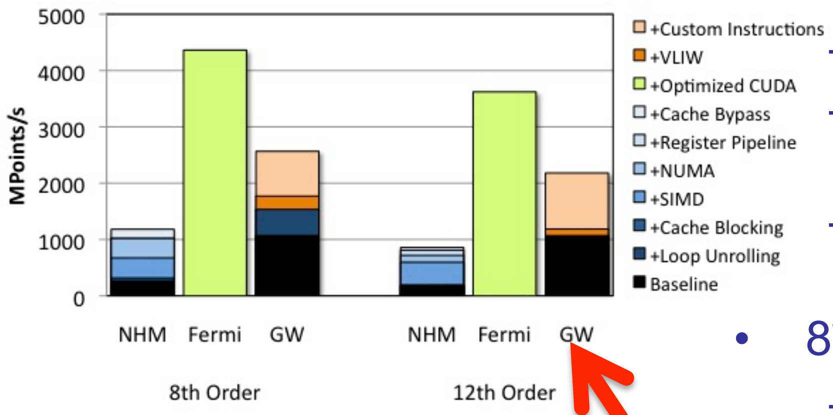


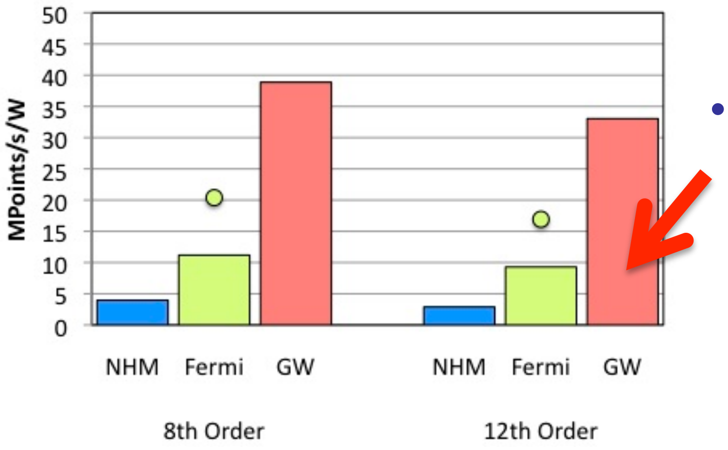
- Developed RTL design for SoC in 45 nm technology using off-the-shelf embedded technology + simulated with RAMP FPGA platform
  - **Tensilica LX2 processor core**
    - off-the-shelf 4-slot SP SIMD, ECC
    - 4-slot VLIW (FLIX)
    - cache hierarchy with Local store + conventional cache.
    - TIE queues interprocessor messaging



  - **NoC fabric for SoC services**
    - 128 cores
    - 4 DDR3 1600 memory controllers
    - 4x 10Gig Ethernet

Example Design Study
Green Wave: Seismic Imaging





- Compare to Nehalem, Fermi
  - All 40-45nm technology
  - Nehalem and Green Wave are 240mm^2 die area with conventional DDR3 memory (same memory perf)
  - Fermi c2050 is 540mm^2 die with DDR5 memory (3x higher memory bandwidth)
- 8th order RTM kernel performance
  - Nehalem auto-tuned to within 15% of theoretical performance limit by Sam Williams
  - Fermi hand-tuned by Paulius Mickavelius of Nvidia
- Off-the-shelf embedded ASIC unable to beat Fermi (but within ~30%)
- However, offers huge gain in energy efficiency
  - Fermi burdened by host (green circle is if you had Fermi without host)
  - Green wave also has advantage of not including anything you don't need for RTM
    - Fermi and Nehalem have to include a lot of extra stuff for other markets such as Graphics.
    - Nehalem also must maintain legacy binary compatibility

Isn’t it too time consuming to design a complete system?

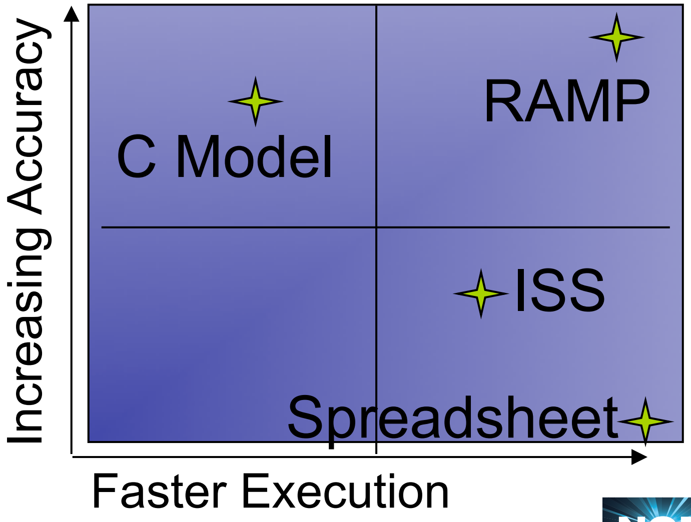
Rapid Prototyping of System Design

Using RAMP to Accelerate the hardware/software co-design cycle

Advanced Hardware Simulation (RAMP)

Enabling Hardware/Software/Science Co-Design

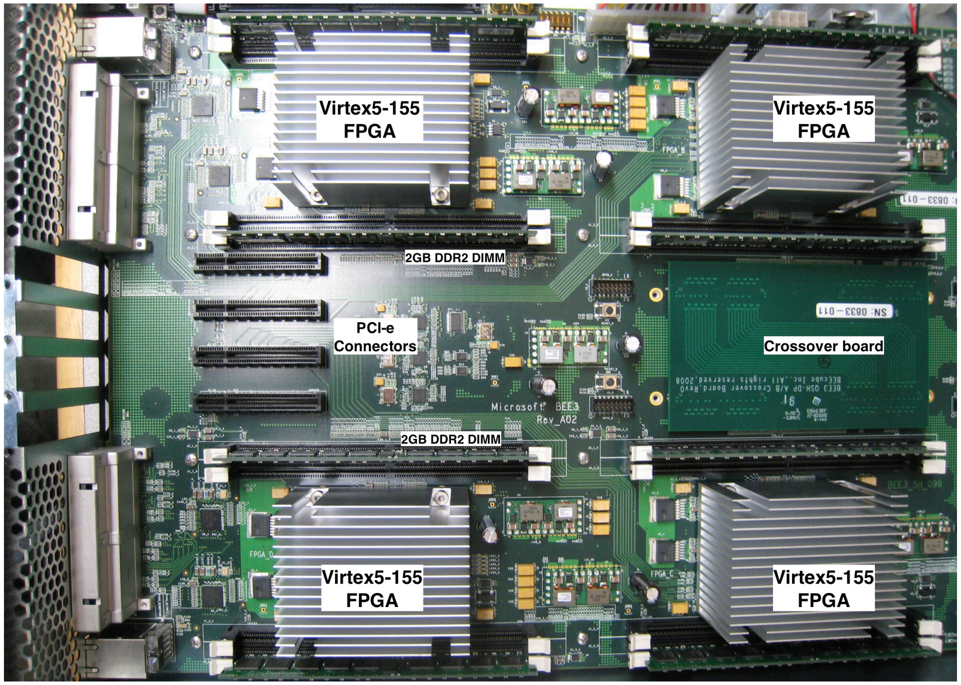
- Research Accelerator for Multi-Processors
  - Simulate hardware *before* it is built!
  - Break slow feedback loop for system designs
  - Enables tightly coupled hardware/software/science co-design *(not possible using conventional approach)*
  - Allows fast performance validation
  - Emulates entire application (not just kernel)

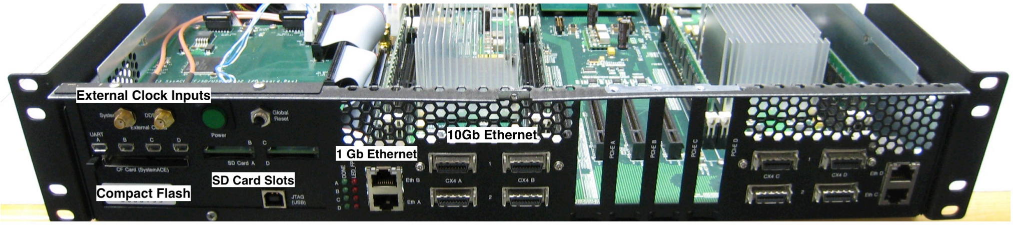


BEE3: Berkeley Emulation Engine

FPGA Platform for Hardware Emulation

- Includes:
  - 4 Xilinx V5-155T FPGAs
  - Up to 8GB DDR2 per FPGA
  - Ring topology connecting FPGAs in combination with a crossover allows for all - to -all connectivity
  - 10Gb connections for inter-board communication
  - 1 Gb Ethernet, PCI-e and UART available for host communication
  - Commercially available from BeeCube



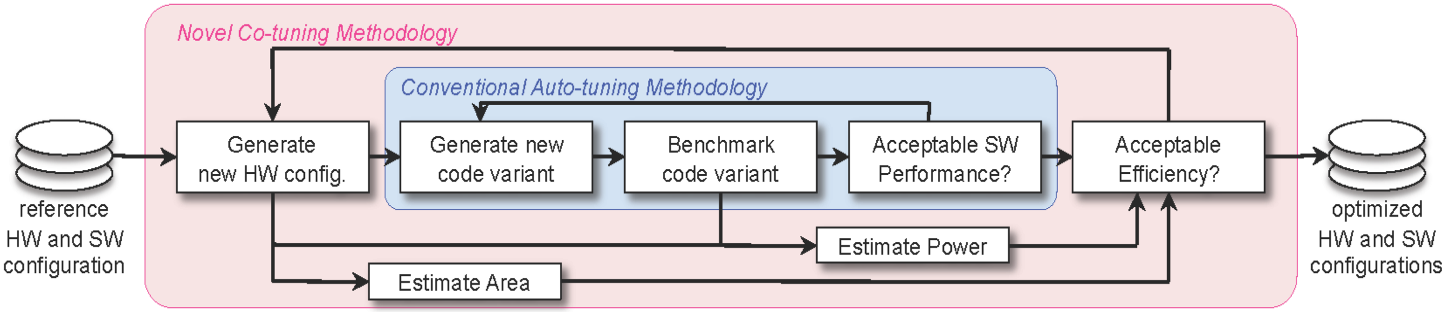


Tuning Hardware to Fit the Problem

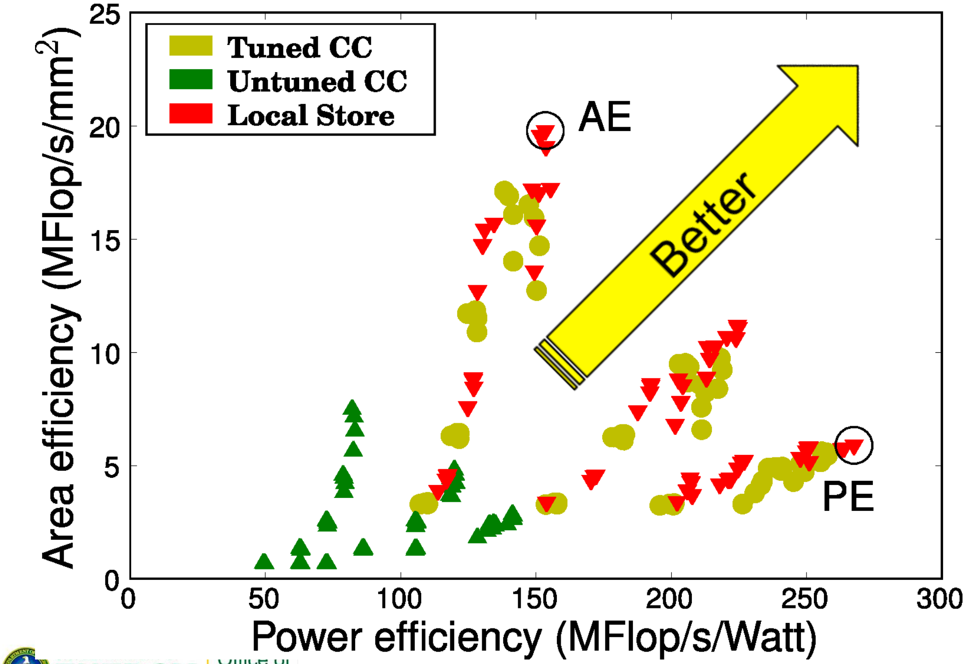
- Software Design Space Exploration: “auto-tuning”
  - Auto-search through parameter space of code optimizations
  - Tune to diverse & complex hardware
- Hardware Design Space Exploration:
  - What if hardware configuration was also parameterized?
  - Search through diverse space of hardware configurations
- What if you could do both together?
  - Auto-tune software for hardware
  - Auto-tune hardware for software
  - Repeat?
- Hardware/Software co-design
  - Demonstrate how to apply to HPC
  - Enable Energy Efficient computing for Extreme Scale Science

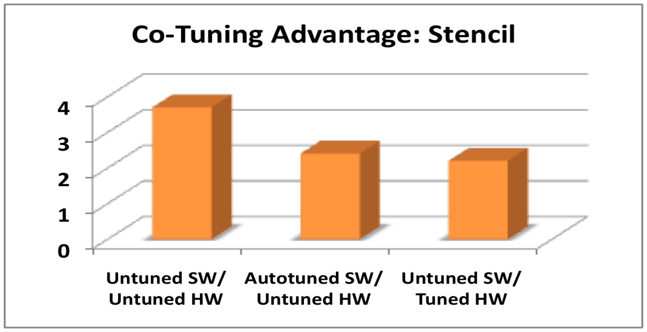
Hardware/Software Co-Tuning for Energy Efficiency

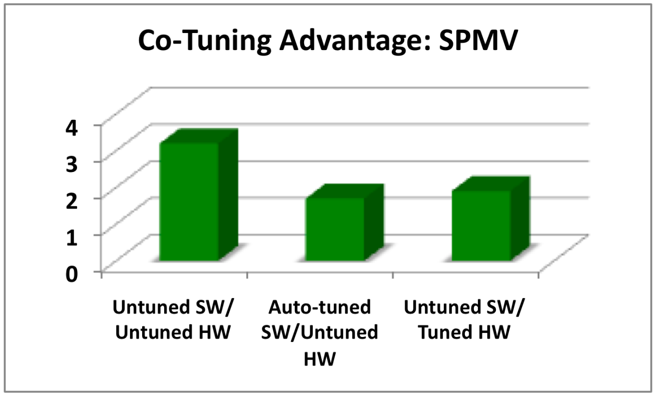
The approach: *Use auto-tuned code when evaluating architecture design points*


Novel Co-tuning Methodology
Conventional Auto-tuning Methodology
reference HW and SW configuration
Generate new HW config.
Generate new code variant
Benchmark code variant
Acceptable SW Performance?
Acceptable Efficiency?
Estimate Power
Estimate Area
optimized HW and SW configurations

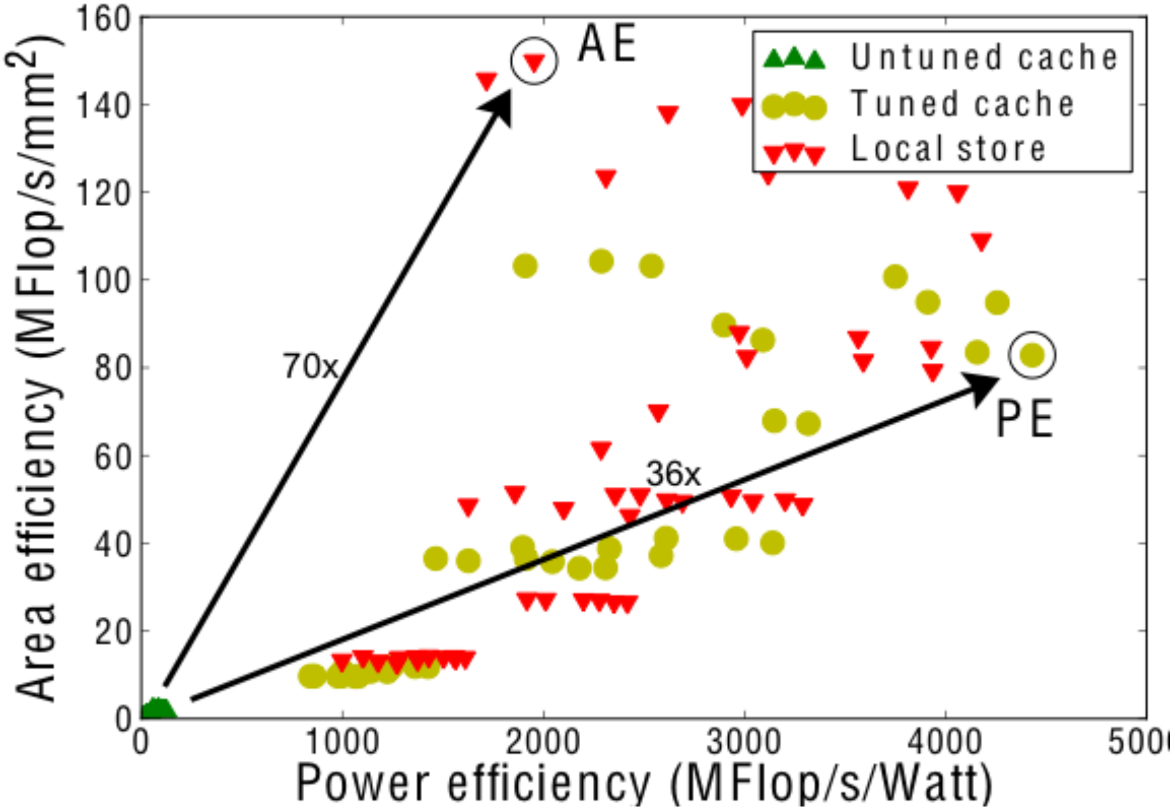
Co-Tuning can improve power-efficiency and area-efficiency by ~4x





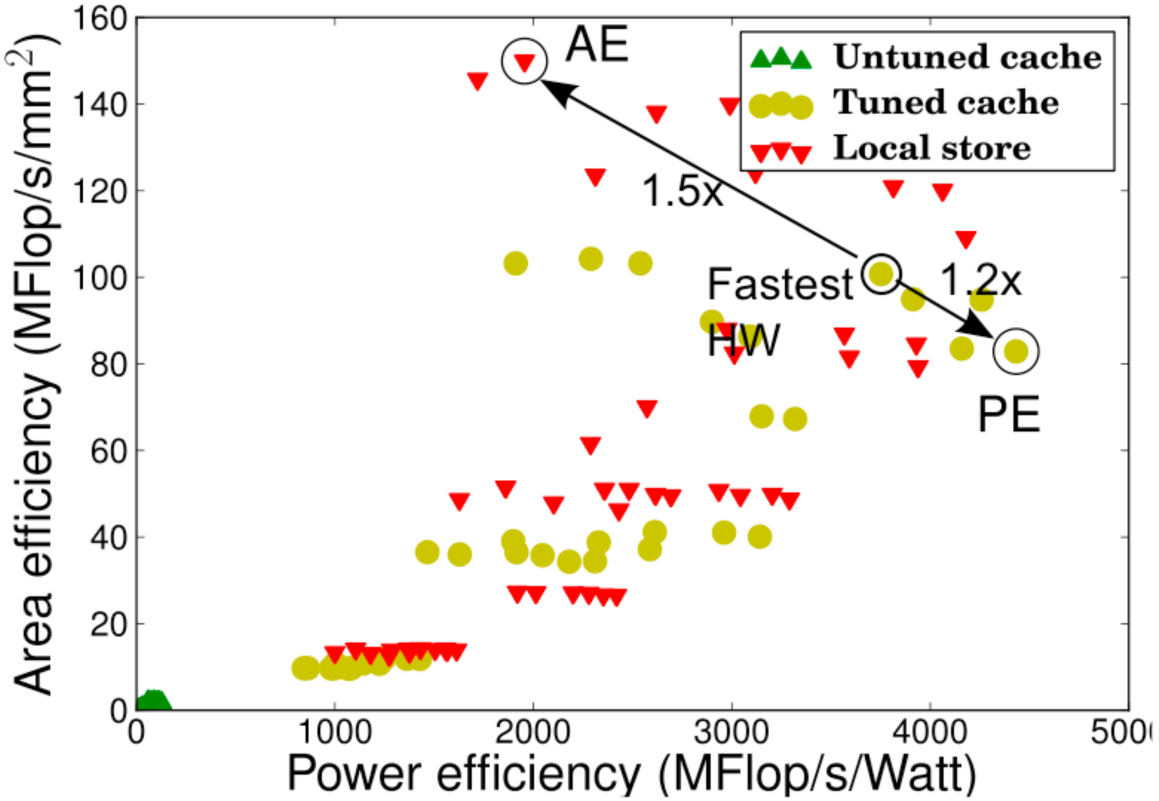


GEMM Co-Design Results



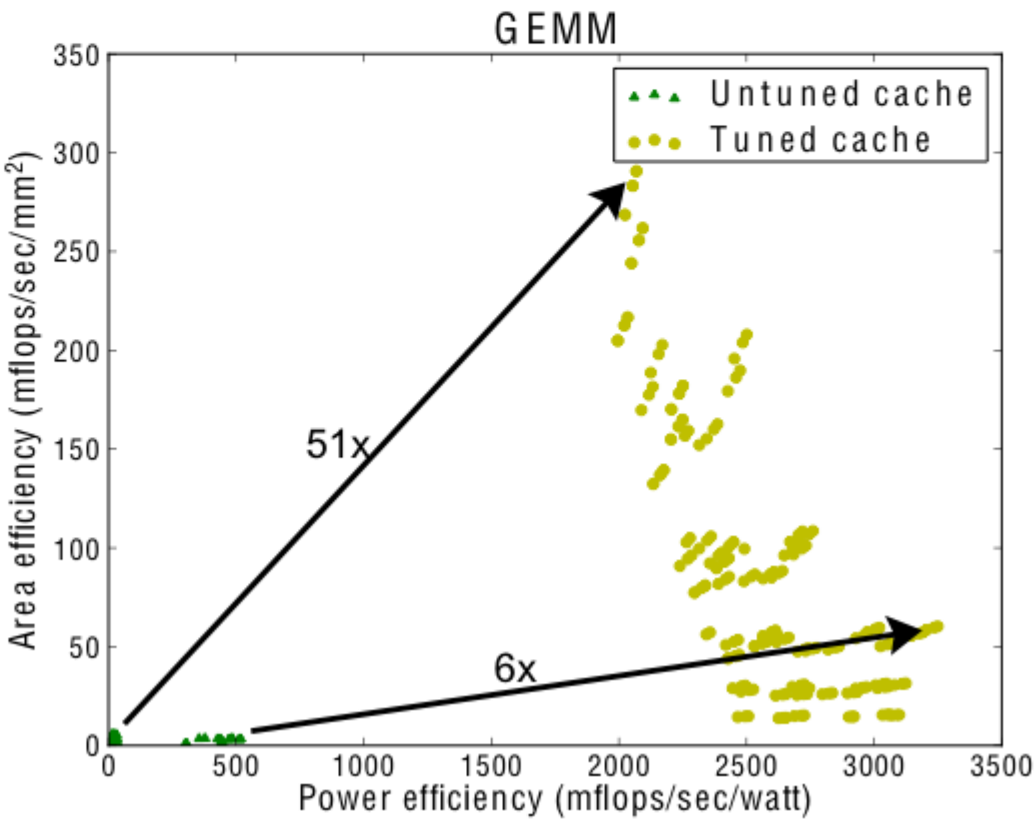
- Each point represents HW design point
  - Best SW performance chosen by autotuner
  - 72 unique configs
  - Runtime: 1 week

GEMM Co-Design Results



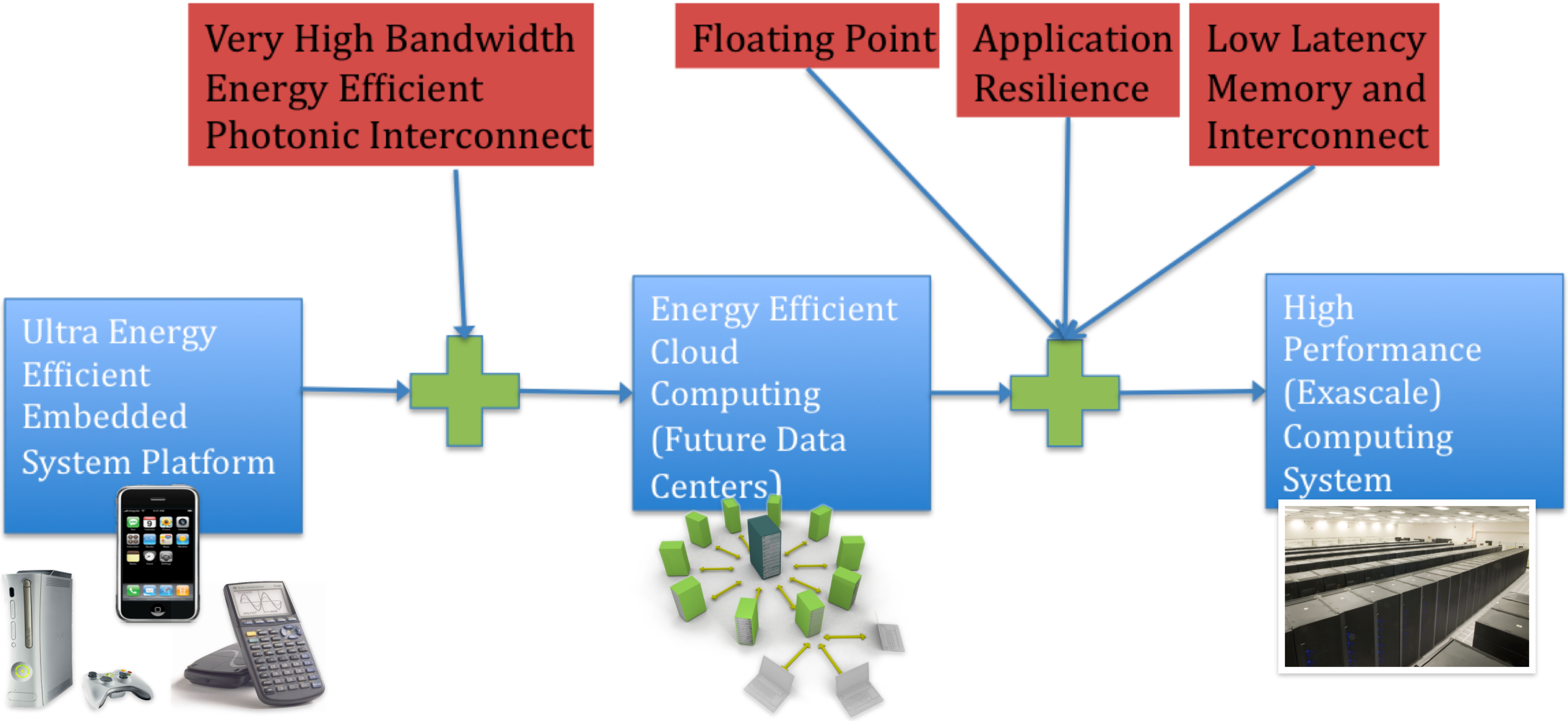
- Each point represents HW design point
  - Best SW performance chosen by autotuner
  - 72 unique configs
  - Runtime: 1 week

GEMM Co-Design Results



- Generated through FPGA Emulation Flow
  - 216 Unique Configs
  - Runtime: hours
  - 125x speedup

Technology Continuity for A Sustainable Hardware Ecosystem



Very High Bandwidth Energy Efficient Photonic Interconnect

Floating Point

Application Resilience

Low Latency Memory and Interconnect

Ultra Energy Efficient Embedded System Platform

Energy Efficient Cloud Computing (Future Data Centers)

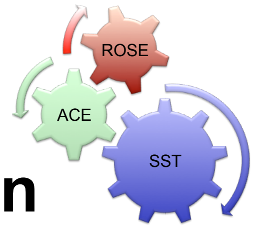
High Performance (Exascale) Computing System

Going further…

Green Flash Impact

- Significant infrastructure development
  - General Stencil autotuning framework
  - FPGA emulation framework
  - Application analysis
- Clear demonstration of improved performance per / watt on multiple scientific codes
  - Future HPC systems are power limited
  - Green Flash methodology provides a path to exascale
- DOE has responded by establishing exascale co-design centers around the nation
  - Green Flash and RAMP have played a key role in affecting this shift
  - DOE funding for ISIS, CoDEX, and application Co-Design centers

CoDEx Overview

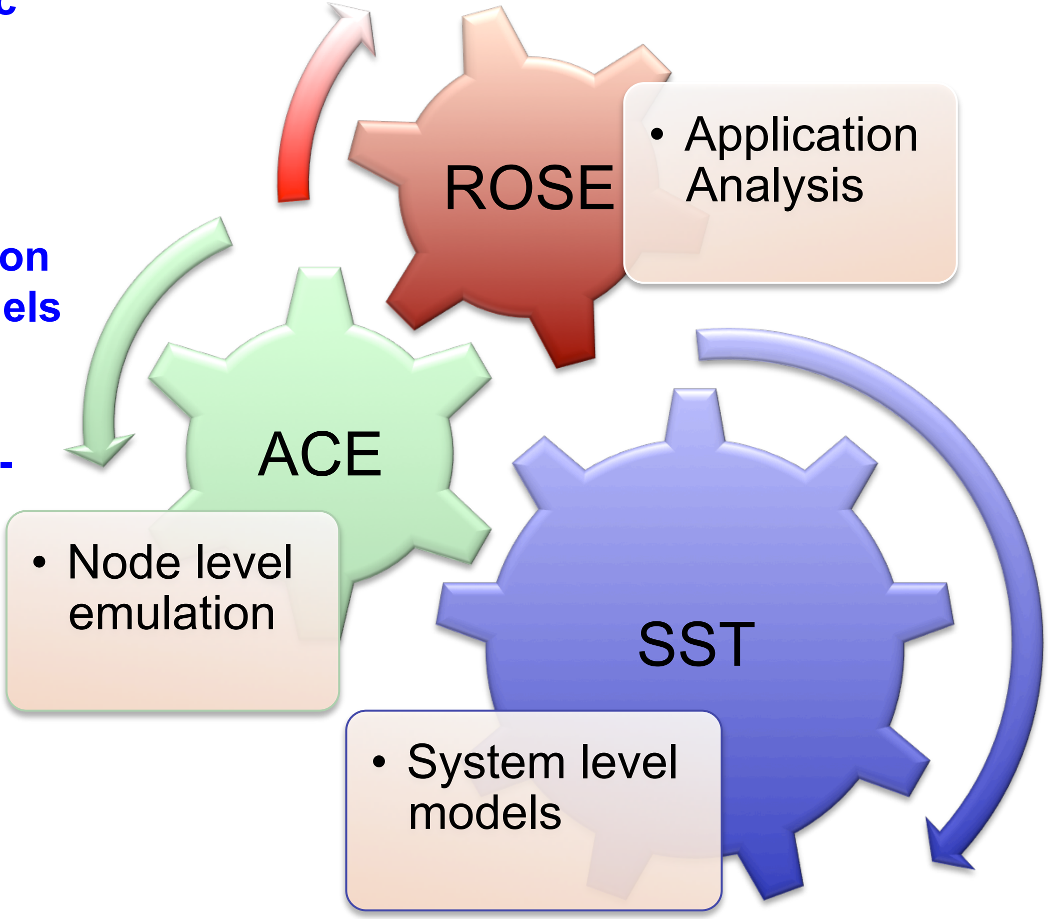


Architectural Simulation to Accelerate CoDesign

ROSE Compiler: **Enables deep analysis of application requirements, semi-automatic generation of skeleton applications, and code generation for ACE and SST.**

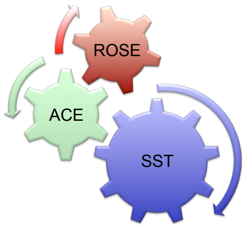
ACE Node Emulation: **Rapid design synthesis and FPGA-accelerated emulation for rapid prototyping cycle accurate models of manycore node designs.**

SST System Simulation: **Enables system-scale simulation through capture of application communication traces and simulation of large-scale interconnects.**

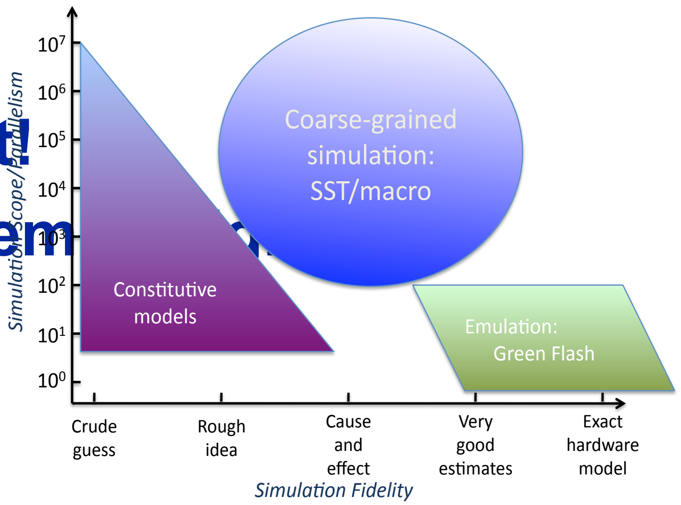


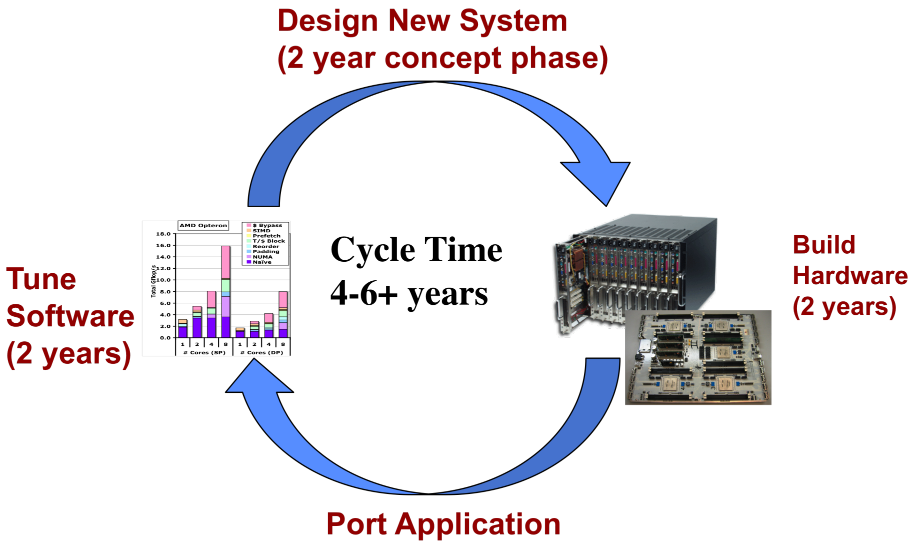
Partnered with multiple DOE CoDesign centers

Role of Architectural Simulation



- Hardware Architectural Simulation
  - Simulate hardware *before* it is built
  - Break slow feedback loop for system
  - Tightly coupled CoDesign process





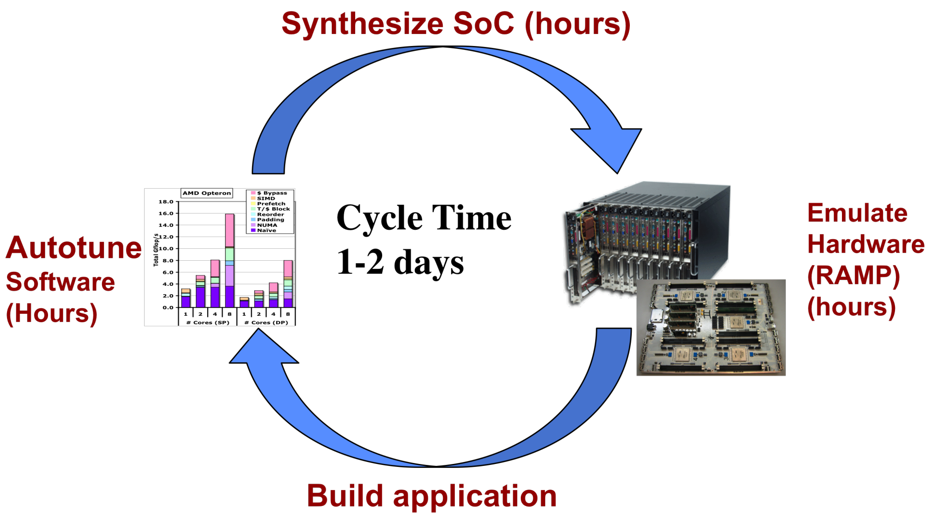
Design New System (2 year concept phase)

Cycle Time 4-6+ years

Build Hardware (2 years)

Port Application

Tune Software (2 years)



Synthesize SoC (hours)

Cycle Time 1-2 days

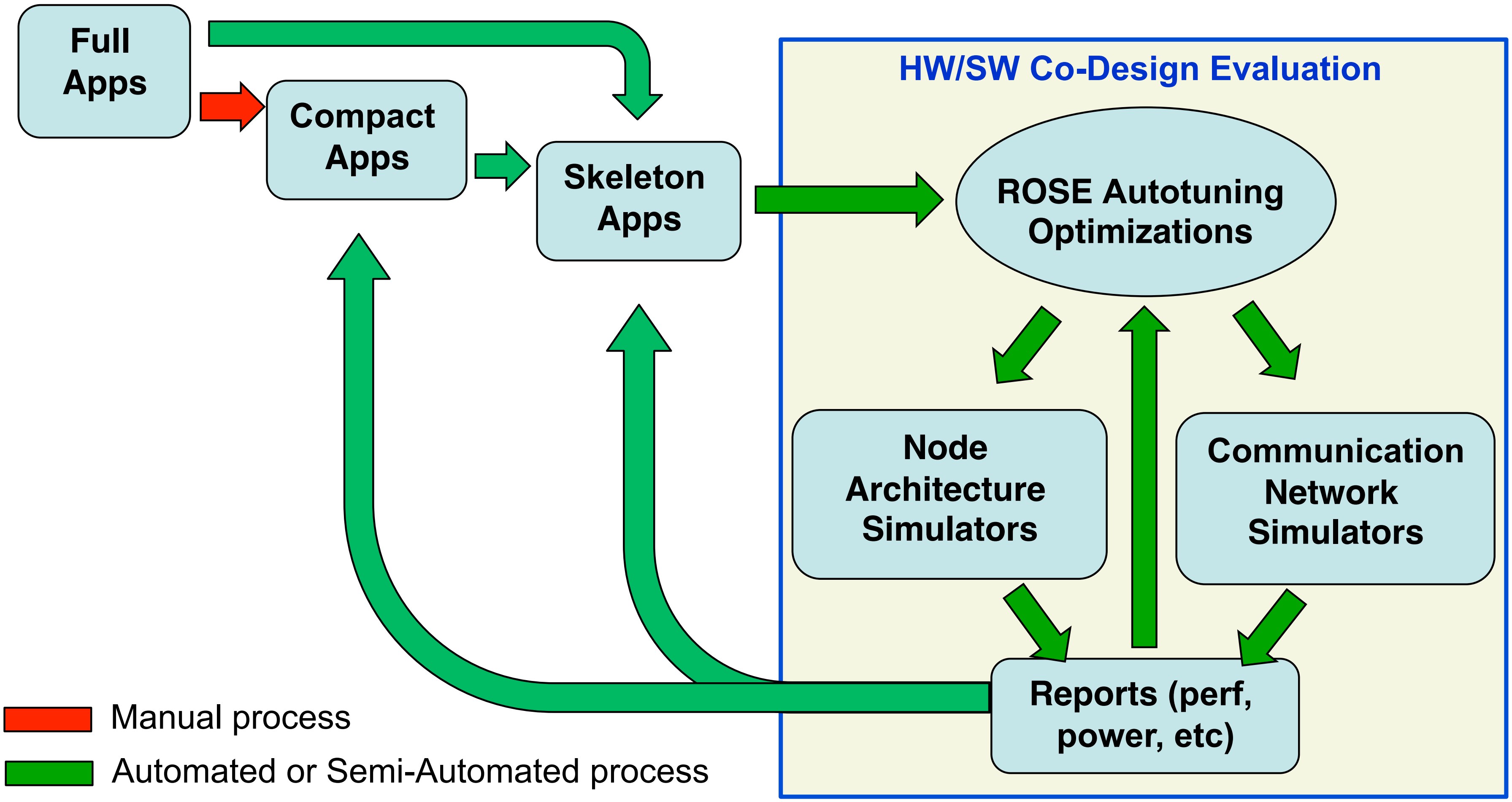
Emulate Hardware (RAMP) (hours)

Build application

Autotune Software (Hours)

CoDesign Tool Flow

Automatic Generation of Skeletons for Rapid Analysis


Full Apps
Compact Apps
Skeleton Apps
HW/SW Co-Design Evaluation
ROSE Autotuning Optimizations
Node Architecture Simulators
Communication Network Simulators
Reports (perf, power, etc)

🟥 Manual process

🟩 Automated or Semi-Automated process

Co-Design is Critical for Exascale

- Major changes ahead for computing, including HPC
  - Resign algorithms to minimize communication (communication-avoiding/optimal)
  - Development of programming models to allow for communication control
  - Feedback to architecture designs
    - How much data-parallelism, local stores, etc.
  - Develop science applications
  - Reduce risk in Exascale program

Acknowledgements

UC Berkeley Students

- Marghoob Mohiyuddin
- Shoaib Kamil
- Jens Krueger
- Kaushik Datta
- Kamesh Madurri
- Cy Chan

Companies

- Tensilica Inc.
- Cray Inc.

LBNL Staff

- David Donofrio
- Leonid Oliker
- Michael Wehner
- Tony Drummond
- Woo-Sun Yang
- Norman Miller
- Sam Williams
- Chuck McParland