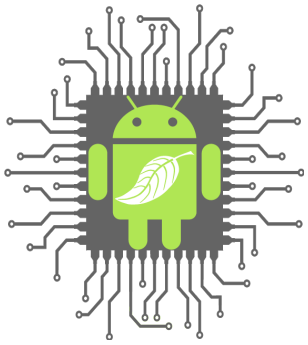
Conservation Cores: Energy-Saving Coprocessors for Nasty Real-World Code

Jack Sampson, Ganesh Venkatesh, Nathan Goulding-Hotta,
Saturnino Garcia, Manish Aurora, Siddhartha Nath, Vikram Bhatt,
Steven Swanson[+] and **Michael Bedford Taylor**[+]



Department of Computer Science and Engineering,
University of California, San Diego

[+]joint project leaders

UCSDCSE
Computer Science and Engineering

This Talk

The Dark Silicon Problem

How to use Dark Silicon to improve energy efficiency (Conservation Cores)

The GreenDroid Mobile Application Processor

Where does dark silicon come from? And how dark is it going to be?

The Utilization Wall:

With each successive process generation, the percentage of a chip that can actively switch drops exponentially due to power constraints.

[Venkatesh, Chakraborty]

We've Hit The Utilization Wall

Utilization Wall: With each successive process generation, the percentage of a chip that can actively switch drops exponentially due to power constraints.

- Scaling theory
  - Transistor and power budgets are no longer balanced
  - Exponentially increasing problem!
- Experimental results
  - Replicated a small datapath
  - More "dark silicon" than active
- Observations in the wild
  - Flat frequency curve
  - "Turbo Mode"
  - Increasing cache/processor ratio

We've Hit The Utilization Wall

Utilization Wall: With each successive process generation, the percentage of a chip that can actively switch drops exponentially due to power constraints.

- Scaling theory
  - Transistor and power budgets are no longer balanced
  - Exponentially increasing problem!
- Experimental results
  - Replicated a small datapath
  - More "dark silicon" than active
- Observations in the wild
  - Flat frequency curve
  - "Turbo Mode"
  - Increasing cache/processor ratio

Classical scaling

Device count	S^2
Device frequency	S
Device cap (power)	1/S
Device V_{dd} (power)	**$1/S^2$**
Utilization ?	

1

Leakage-limited scaling

Device count	S^2
Device frequency	S
Device cap->power	1/S
Device V_{dd} (power)	**~1**
Utilization ?	

S^2

We've Hit The Utilization Wall

Utilization Wall: With each successive process generation, the percentage of a chip that can actively switch drops exponentially due to power constraints.

- Scaling theory
  - Transistor and power budgets are no longer balanced
  - Exponentially increasing problem!
- Experimental results
  - Replicated a small datapath
  - More "dark silicon" than active
- Observations in the wild
  - Flat frequency curve
  - "Turbo Mode"
  - Increasing cache/processor ratio

Classical scaling

Device count	S^2
Device frequency	S
Device cap (power)	1/S
Device V_{dd} (power)	**$1/S^2$**
Utilization ?	**1**

1

Leakage-limited scaling

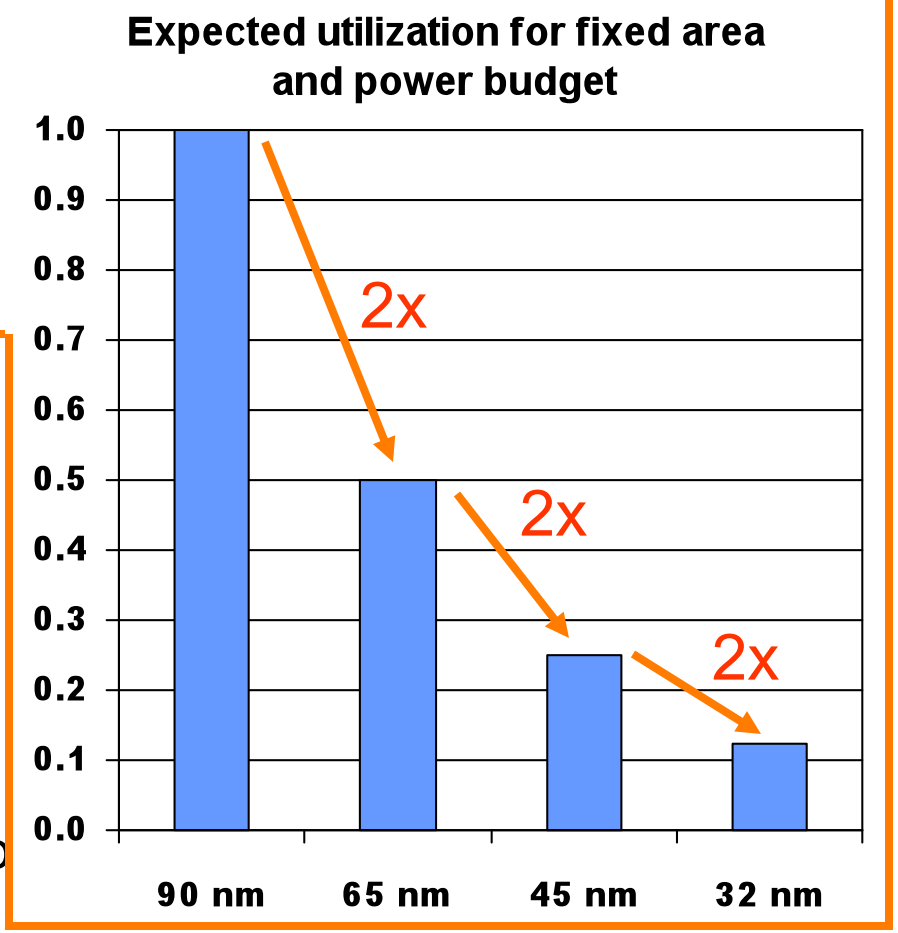
Device count	S^2
Device frequency	S
Device cap->power	1/S
Device V_{dd} (power)	**~1**
Utilization ?	**$1/S^2$**

S^2

We've Hit The Utilization Wall

Utilization Wall: With each successive process generation, the percentage of a chip that can actively switch drops exponentially due to power constraints.

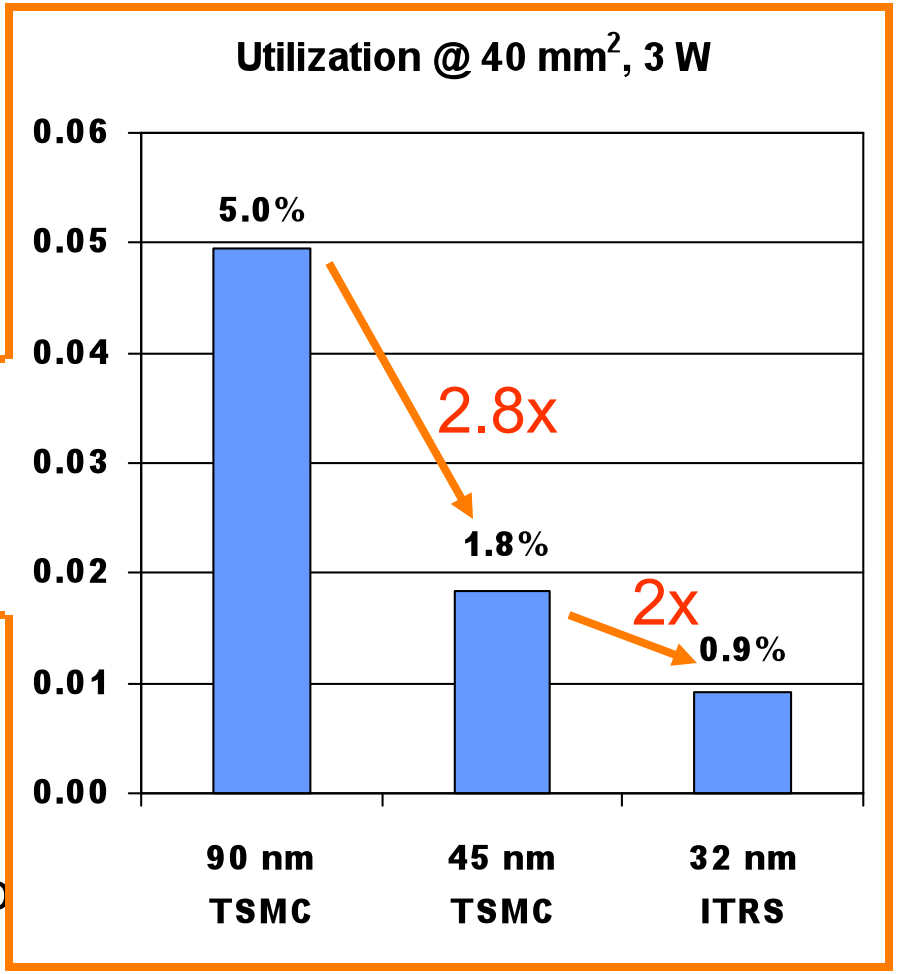
- Scaling theory
  - Transistor and power budgets are no longer balanced
  - Exponentially increasing problem!
- Experimental results
  - Replicated a small datapath
  - More "dark silicon" than active
- Observations in the wild
  - Flat frequency curve
  - "Turbo Mode"
  - Increasing cache/processor ratio



We've Hit The Utilization Wall

Utilization Wall: With each successive process generation, the percentage of a chip that can actively switch drops exponentially due to power constraints.

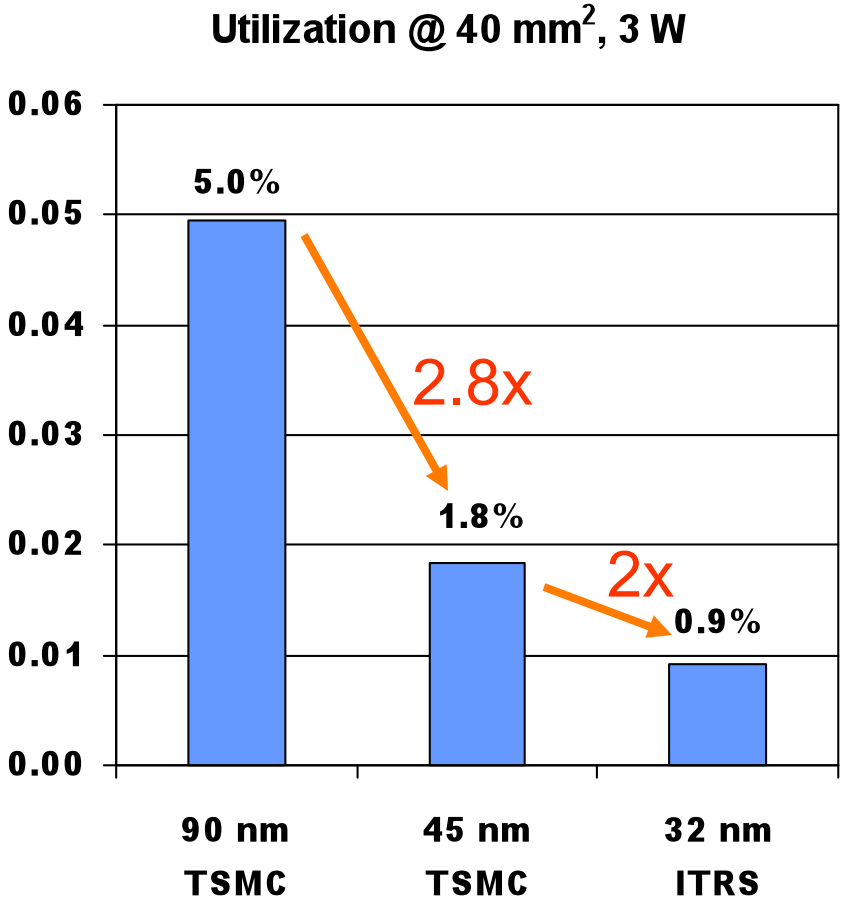
- Scaling theory
  - Transistor and power budgets are no longer balanced
  - Exponentially increasing problem!
- Experimental results
  - Replicated a small datapath
  - More "dark silicon" than active
- Observations in the wild
  - Flat frequency curve
  - "Turbo Mode"
  - Increasing cache/processor ratio



We've Hit The Utilization Wall

Utilization Wall: With each successive process generation, the percentage of a chip that can actively switch drops exponentially due to power constraints.

- Scaling theory
  - Transistor and power budgets are no longer balanced
  - Exponentially increasing problem!
- Experimental results
  - Replicated a small datapath
  - More "dark silicon" than active
- Observations in the wild
  - Flat frequency curve
  - "Turbo Mode"
  - Increasing cache/processor ratio

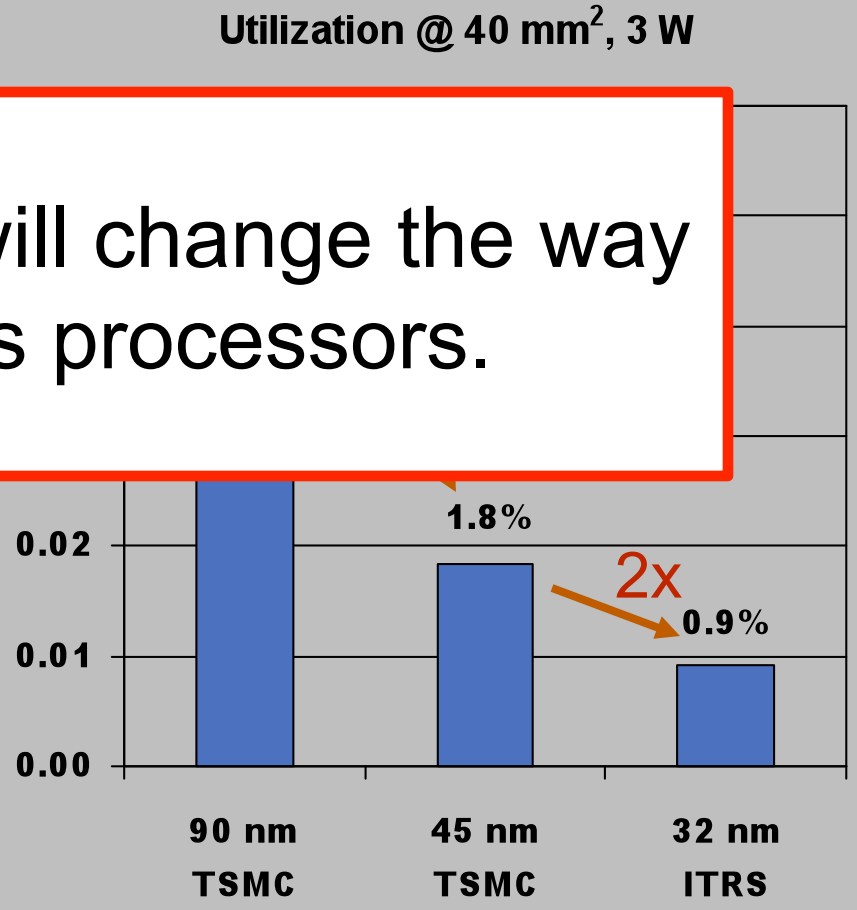


We've Hit The Utilization Wall

Utilization Wall: With each successive process generation, the percentage of a chip that can actively switch drops exponentially due to power constraints.

- Scaling theory
  - –
  - –
- E
  - –
  - – More "dark silicon" than active
- Observations in the wild
  - – Flat frequency curve
  - – "Turbo Mode"
  - – Increasing cache/processor ratio

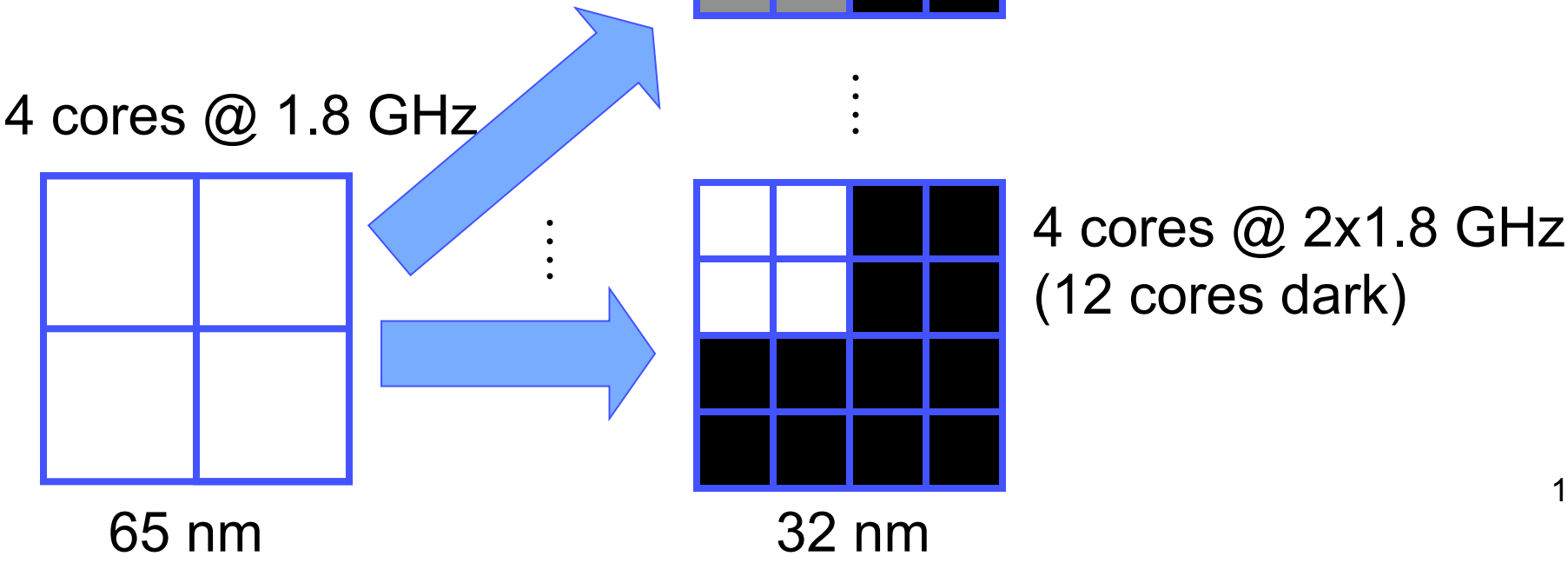
The utilization wall will change the way everyone builds processors.



Utilization Wall: Dark Silicon Leads to Dark Implications for Multicore

Spectrum of tradeoffs between # of cores and frequency

Example:
65 nm → 32 nm (S = 2)



This Talk

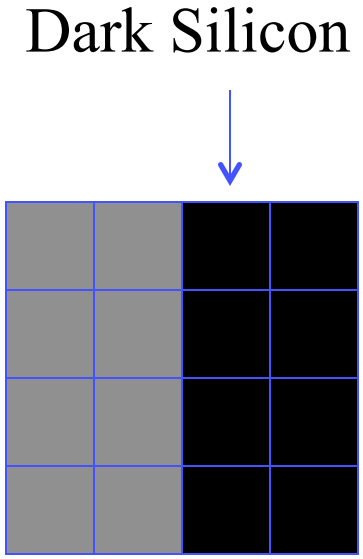
The Dark Silicon Problem

How to use Dark Silicon to improve energy efficiency (Conservation Cores)

The GreenDroid Mobile Application Processor

What do we do with Dark Silicon?

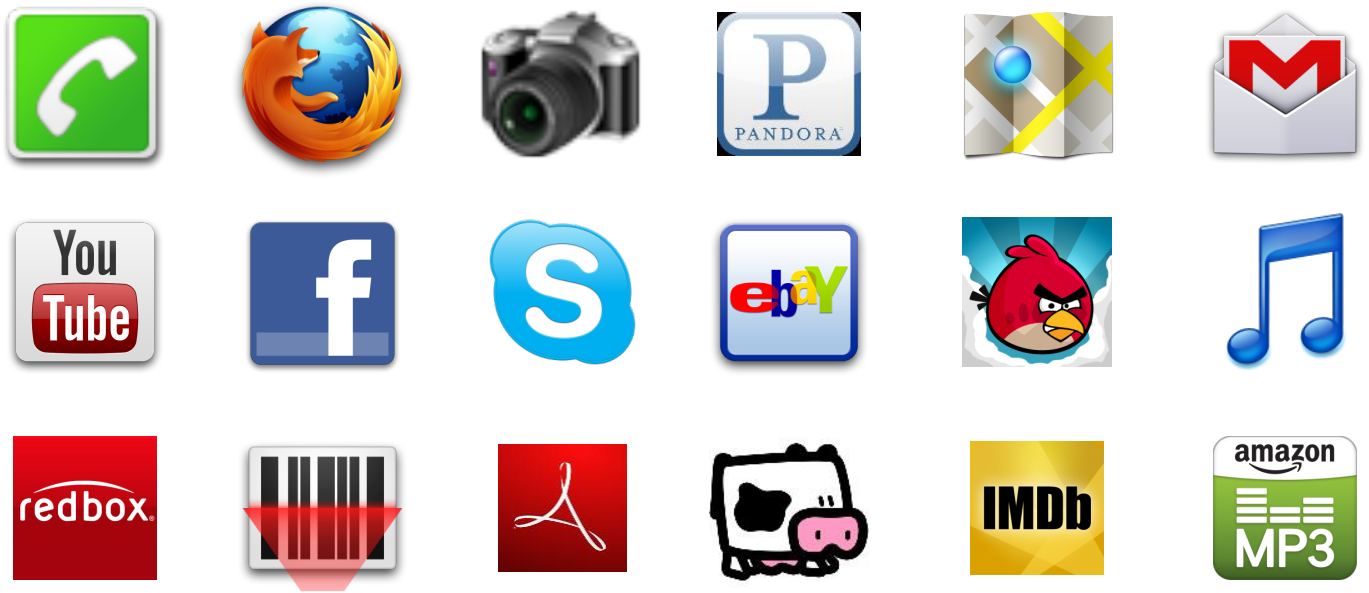
- Idea: Leverage dark silicon to “fight” the utilization wall



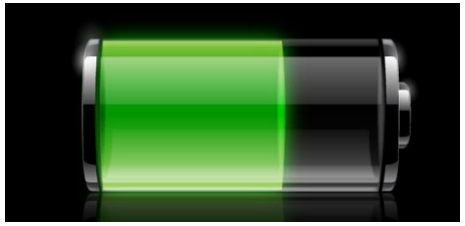
- Insights:
  - Power is now more expensive than area
  - Specialized logic has been shown as an effective way to improve energy efficiency (10-1000x)
- Our Approach:
  - Fill dark silicon with specialized energy-saving coprocessors that save energy on common apps
  - Only turn on the cores as you need them
  - Power savings can be applied to other programs, increasing throughput
- *Energy saving coprocessors provide an architectural way to trade area for an effective increase in power budget!*

Example: Today's smartphone

- Consumers demand rich functionality (desktop → mobile)



- Utilization Wall still applies
  - Active Power budget is set by
    (battery capacity) / (# hrs active use between recharges)
    rather than thermal design point.
- Still need to reduce computation energy



Using accelerators to reduce energy

- Smartphones already combine a general-purpose processor with specialized coprocessors known as *accelerators*

 Audio  Video  Images  Graphics

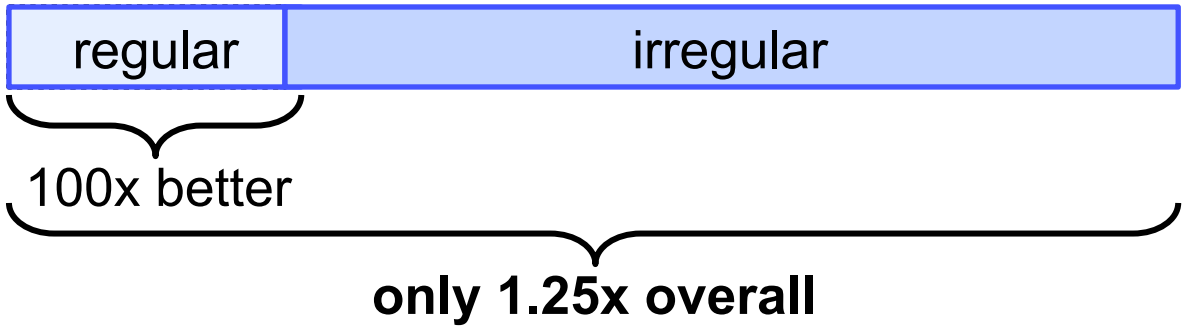
- Accelerators usually speed up computation *and* reduce energy

- Accelerators exist for “easy-to-parallelize”, or **regular**, code
  - Well-structured
  - Moderate or High Parallelism
  - Predictable memory accesses and branch directions
  - Relatively small # of lines of code
  - Often requires human guidance to create (#pragmas or worse)

But what about irregular code?

- Many applications use irregular, difficult-to-parallelize code, for which no accelerators exist

- Amdahl's Law: Overall energy efficiency *depends on the fraction of the total code that is optimized!*

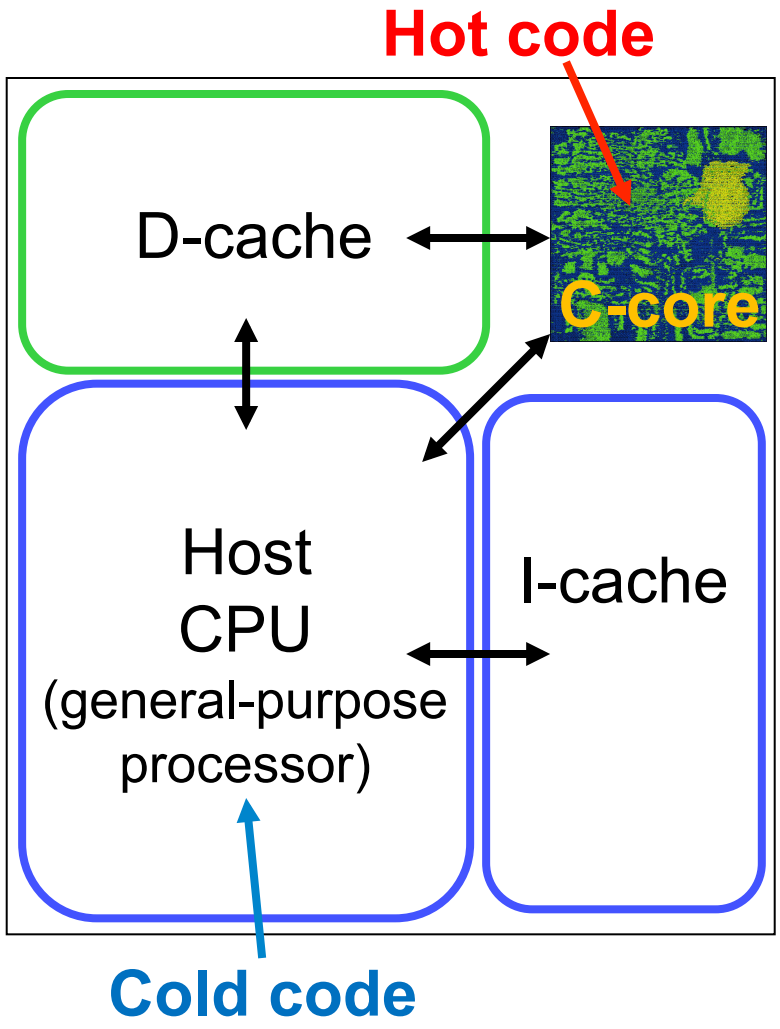


- To gain large energy savings through specialization:
  - We need energy-saving coprocessors that target irregular code, and
  - We need many, many such coprocessors to get high coverage
    - need to solve both design effort and architectural scalability problems

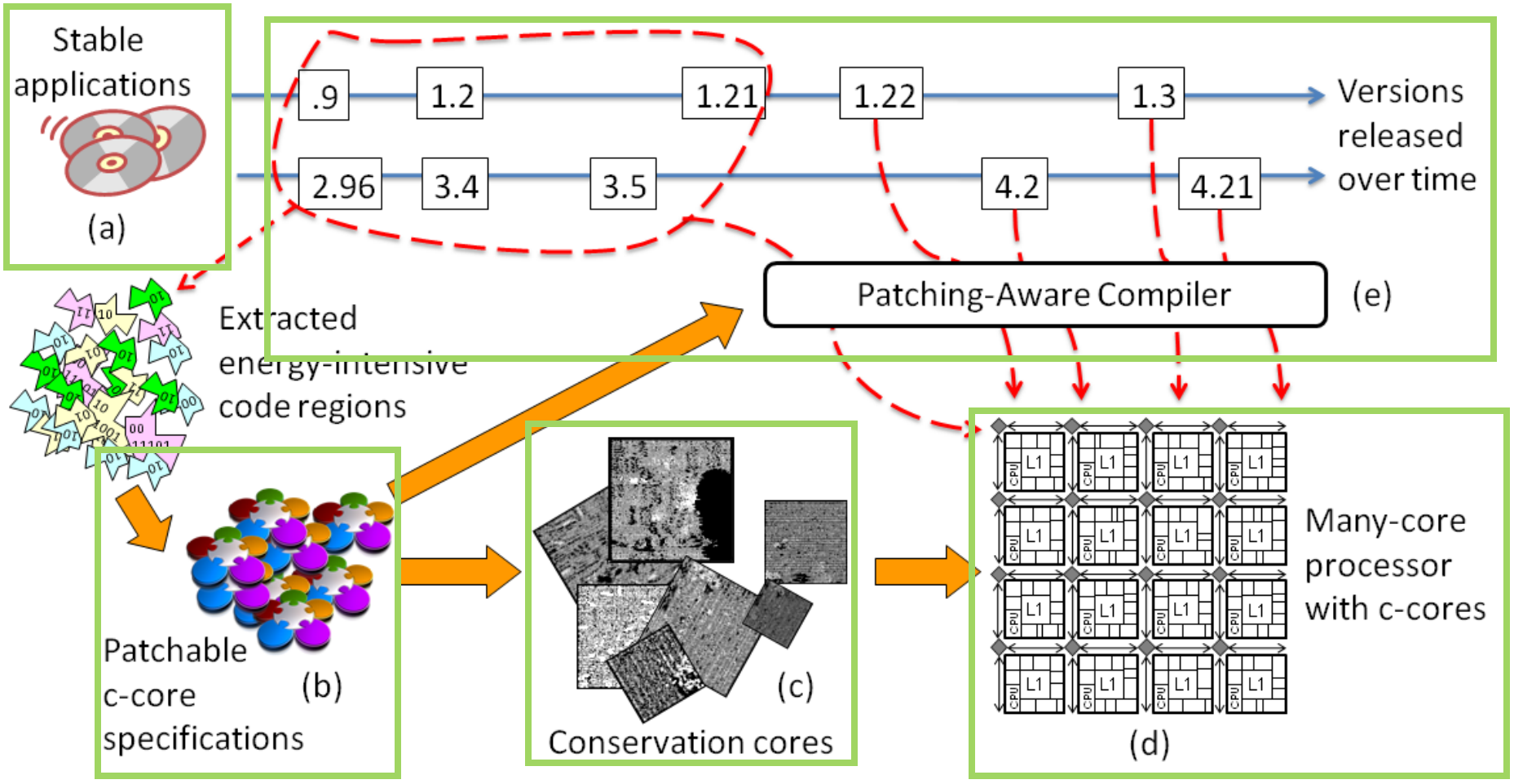
Conservation Cores (C-cores)

"Conservation Cores: Reducing the Energy of Mature Computations," Venkatesh et al., ASPLOS '10

- Specialized coprocessors for reducing energy in irregular code
  - Hot code implemented by c-cores, cold code runs on host CPU;
  - Shared D-cache
  - Patching support for hardware
- Fully-automated toolchain
  - **No "deep" analysis or transformations required**
  - C-cores automatically generated from hot program regions
  - Drop-in replacements for code
  - Design-time scalable
- Energy-efficient
  - Up to 18x for targeted hot code



The C-core Life Cycle



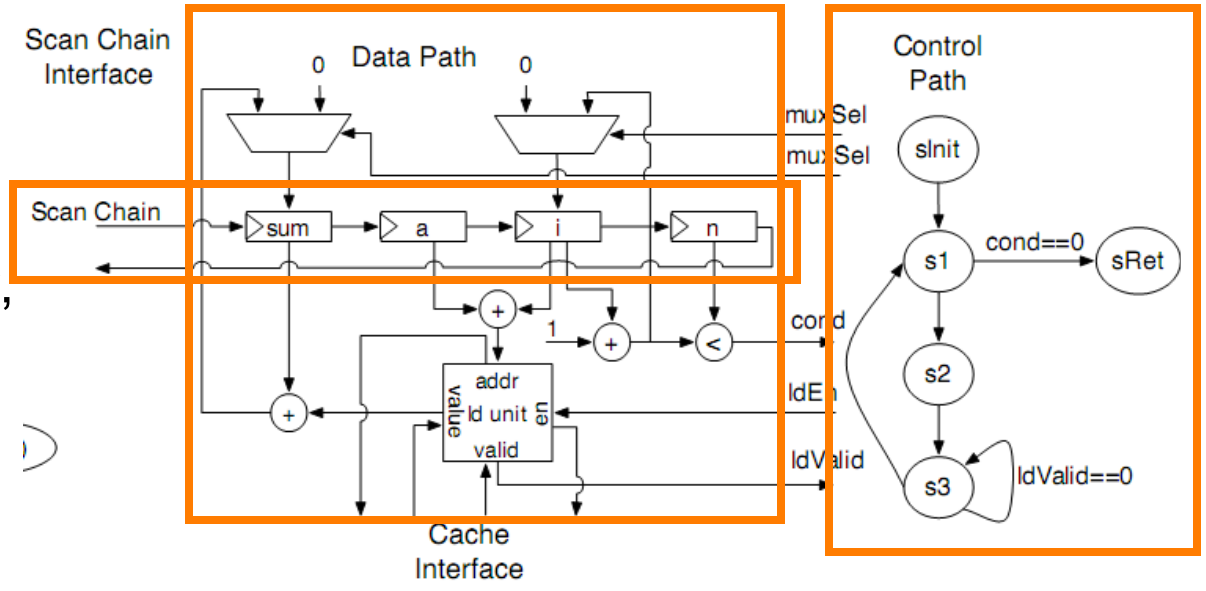
Constructing a C-core

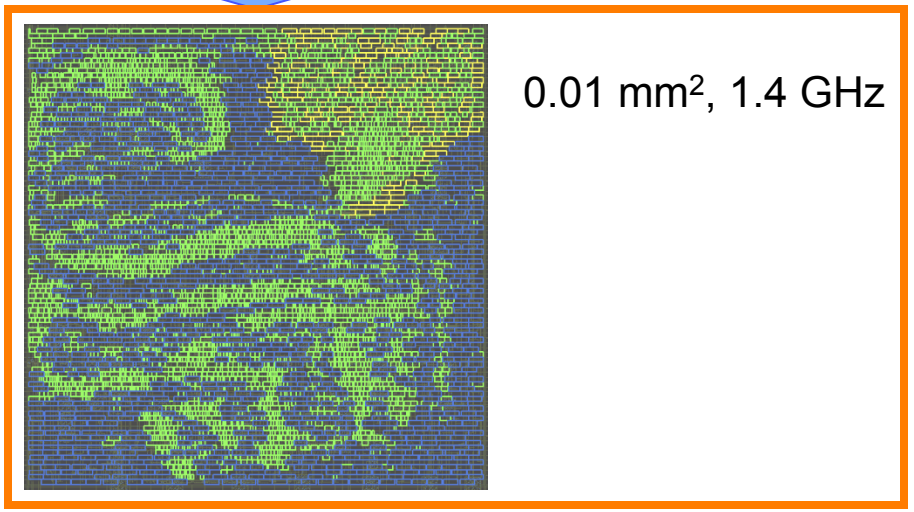
- C-cores start with source code
  - Irregular or regular programs
  - Parallelism-agnostic
- Supports essentially all of C:
  - Complex control flow
    e.g., goto, switch, function calls
  - Arbitrary memory structures
    e.g., pointers, structs, stack, heap
  - Arbitrary operators
    e.g., floating point, divide
  - Memory coherent with host CPU

```
sumArray(int *a, int n)
{
  int i = 0;
  int sum = 0;

  for (i = 0; i < n; i++) {
    sum += a[i];
  }

  return sum;
}
```

Constructing a C-core

- Compilation
  - C-core selection
  - SSA, infinite register, 3-address code
  - Direct mapping from CFG and DFG
  - Scan chain insertion
- Verilog → Synthesis, P&R
  - 45 nm process
  - Synopsys CAD flow
    - Synthesis
    - Placement
    - Clock tree generation
    - Routing



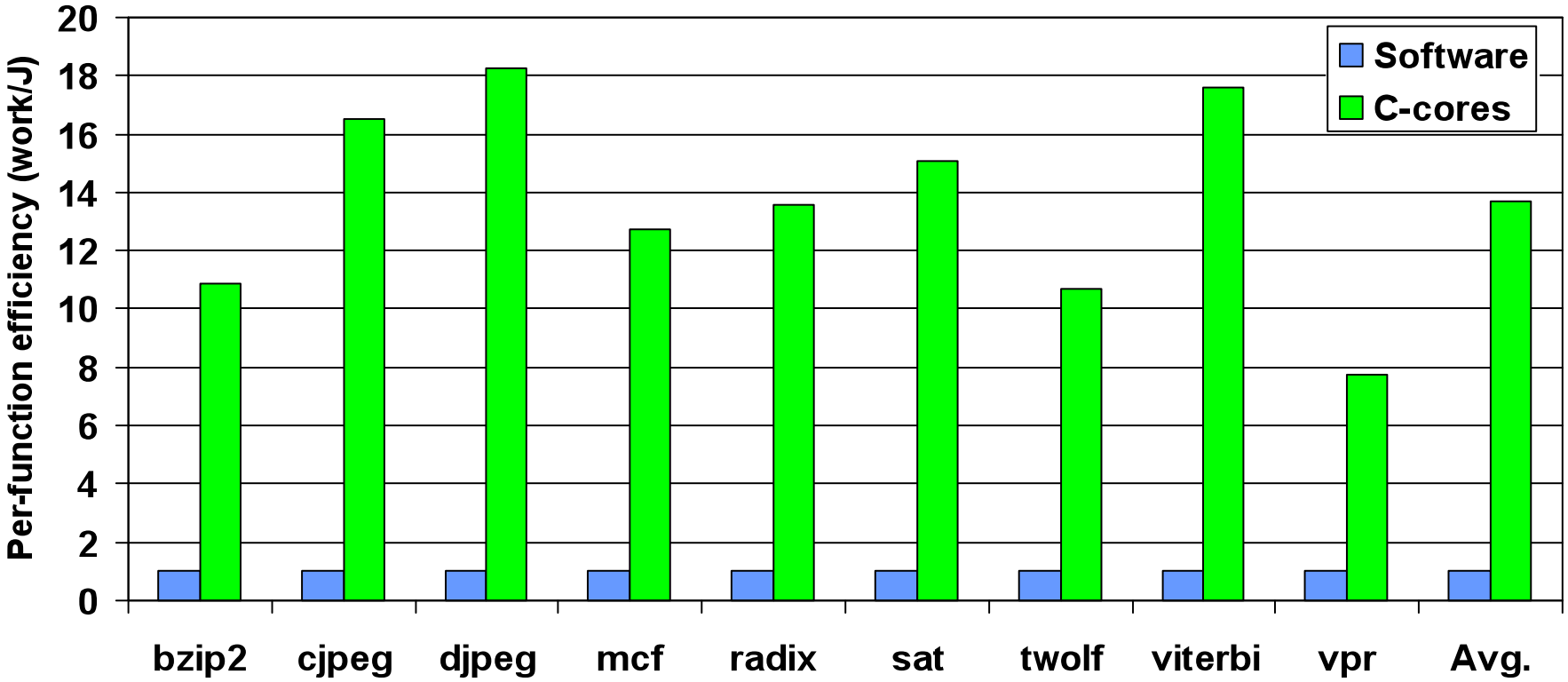


C-cores Experimental Data

- We automatically built 21 c-cores for 9 "hard" applications
  - 45 nm TSMC
  - Vary in size from 0.10 to 0.25 mm^2
  - Frequencies from 1.0 to 1.4 GHz

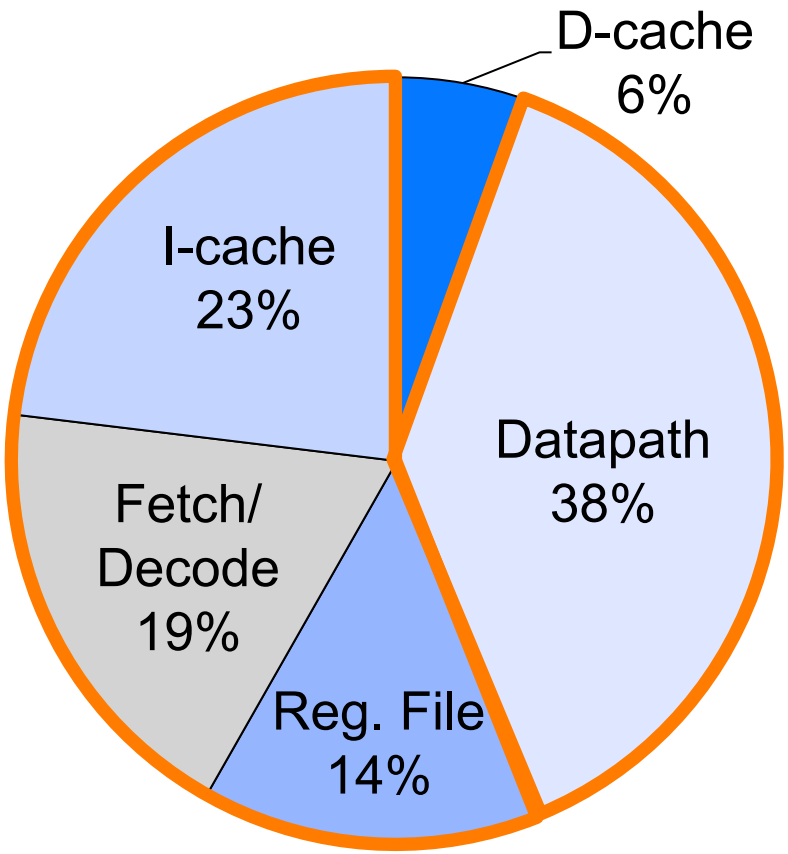
Application	# C-cores	Area (mm^2)	Frequency (MHz)
bzip2	1	0.18	1235
cjpeg	3	0.18	1451
djpeg	3	0.21	1460
mcf	3	0.17	1407
radix	1	0.10	1364
sat solver	2	0.20	1275
twolf	6	0.25	1426
viterbi	1	0.12	1264
vpr	1	0.24	1074

C-core Energy Efficiency: Non-cache Operations

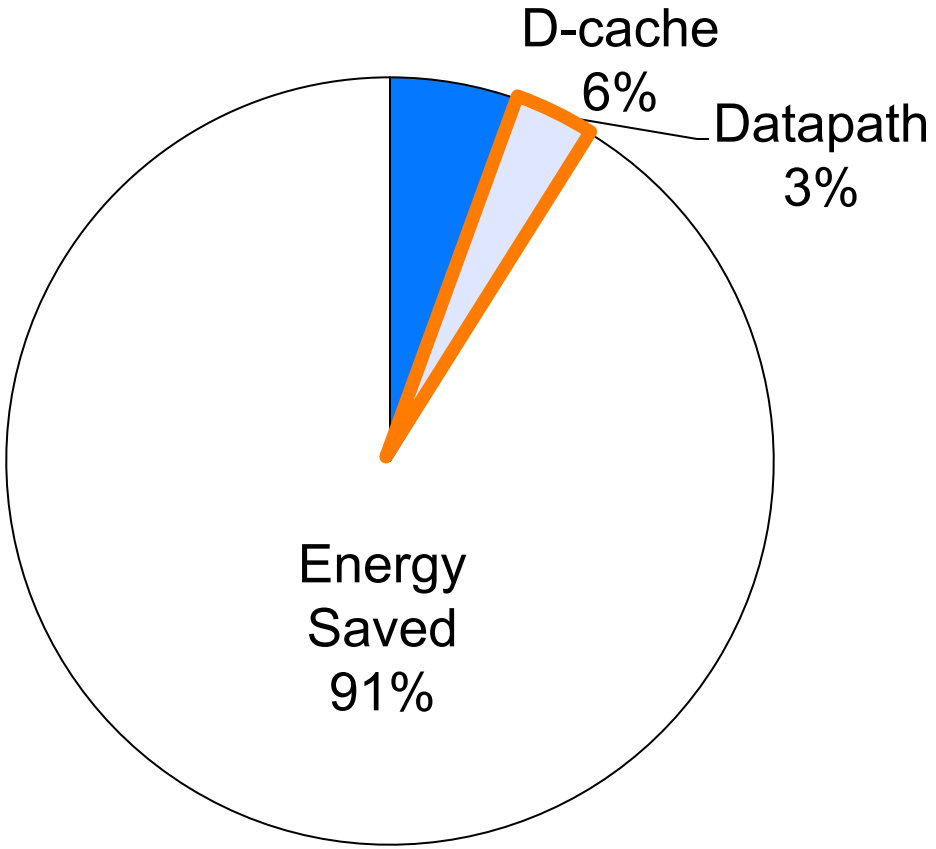


- Up to 18x more energy-efficient (13.7x on average), compared to running on an efficient MIPS processor
- Performance is also better (e.g. 1.3x)

Where do the energy savings come from?



MIPS baseline
91 pJ/instr.



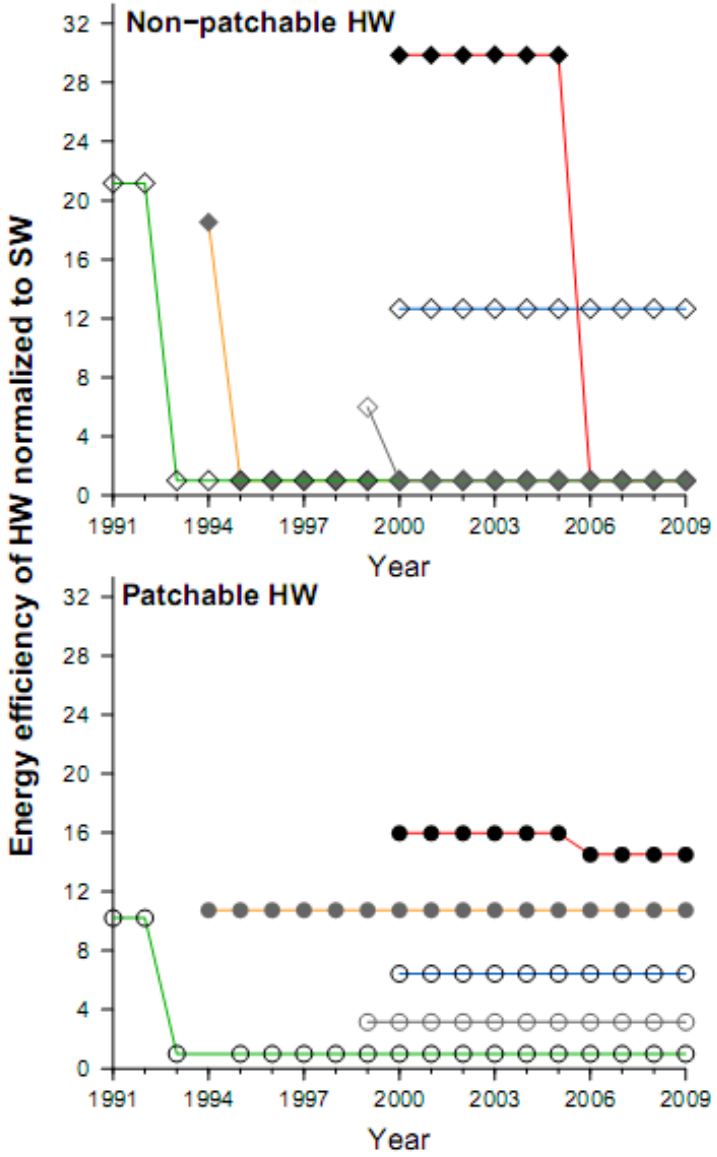
C-cores
8 pJ/instr.

Supporting Software Changes

- Software may change – HW must remain usable
  - C-cores unaffected by changes to cold regions
- Can support any changes, through *patching*
  - Arbitrary insertion of code – software exception mechanism
  - Changes to program constants – configurable registers
  - Changes to operators – configurable functional units
- Software exception mechanism
  - Scan in values from c-core
  - Execute in processor
  - Scan out values back to c-core to resume execution

Patchability Payoff: Longevity

- Graceful degradation
  - Lower initial efficiency
  - Much longer useful lifetime
- Increased viability
  - With patching, utility lasts ~10 years for 4 out of 5 applications
  - Decreases risks of specialization



This Talk

The Dark Silicon Problem

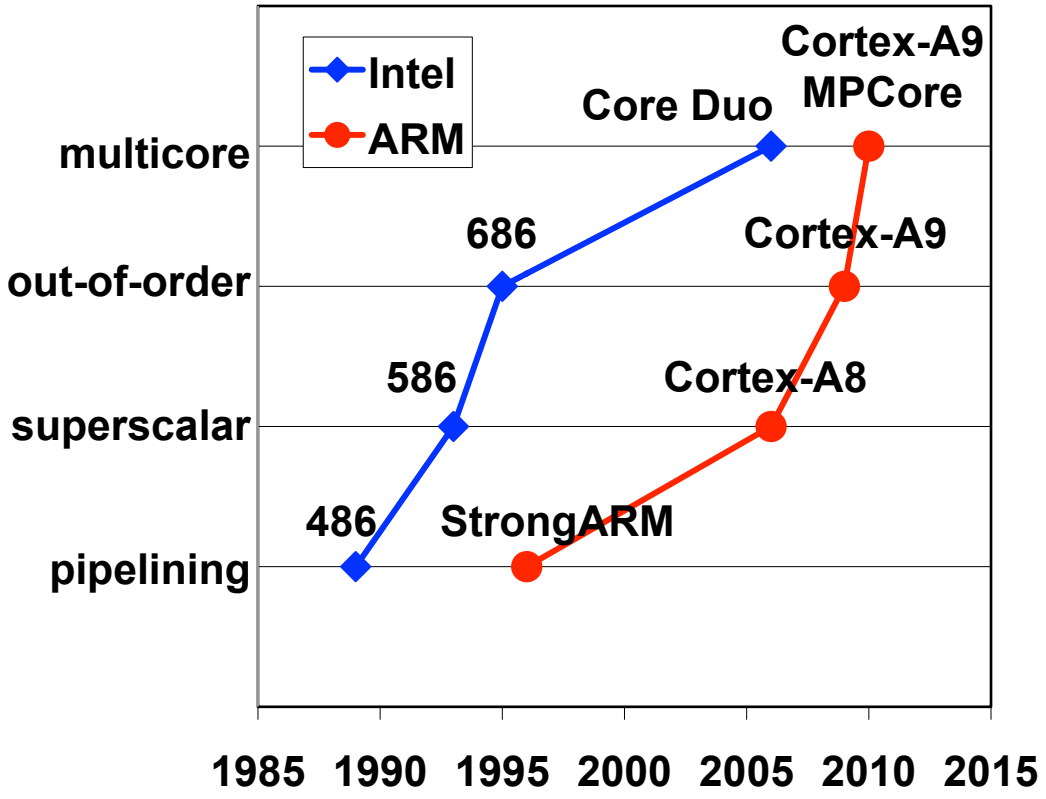
How to use Dark Silicon to improve energy efficiency

The GreenDroid Mobile Application Processor

Mobile Application Processors Face the Utilization Wall

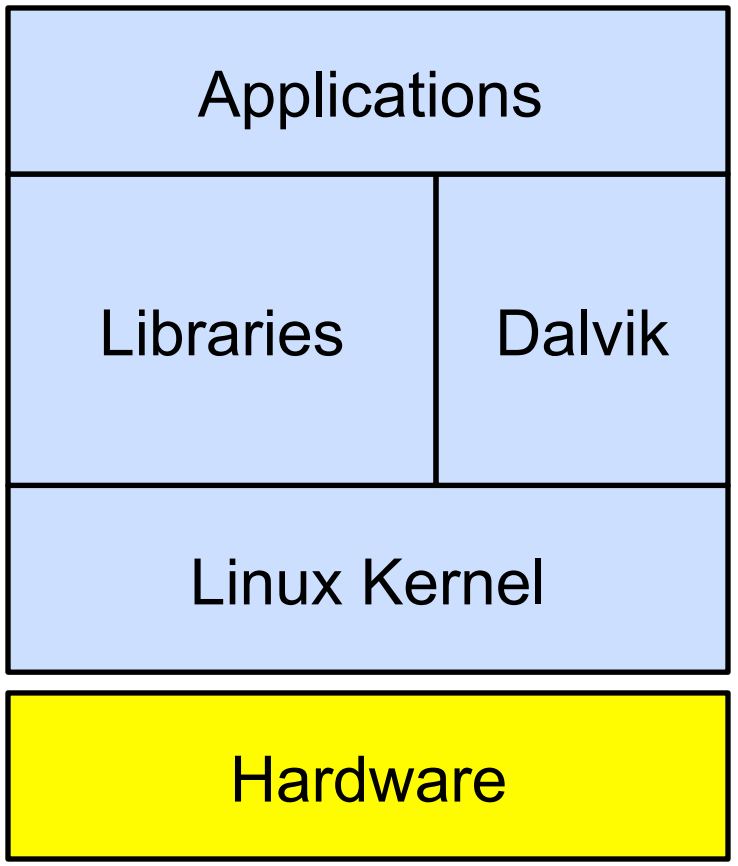
- The evolution of mobile application processors mirrors that of microprocessors

- Application processors face the utilization wall
  - Growing performance demands
  - Extreme energy/power constraints
    (mostly battery)



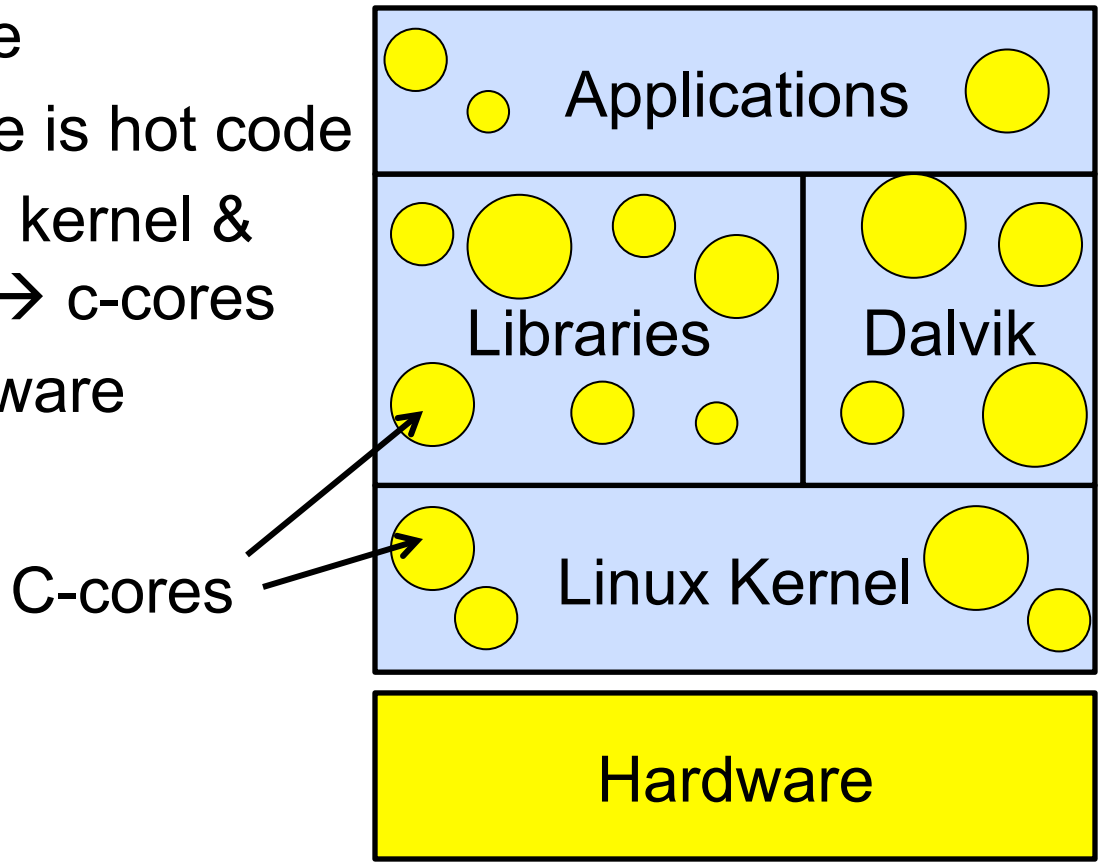
Android™

- Google's OS + app. environment for mobile devices
- Java applications run on the Dalvik virtual machine
- Apps share a set of libraries (libc, OpenGL, SQLite, etc.)



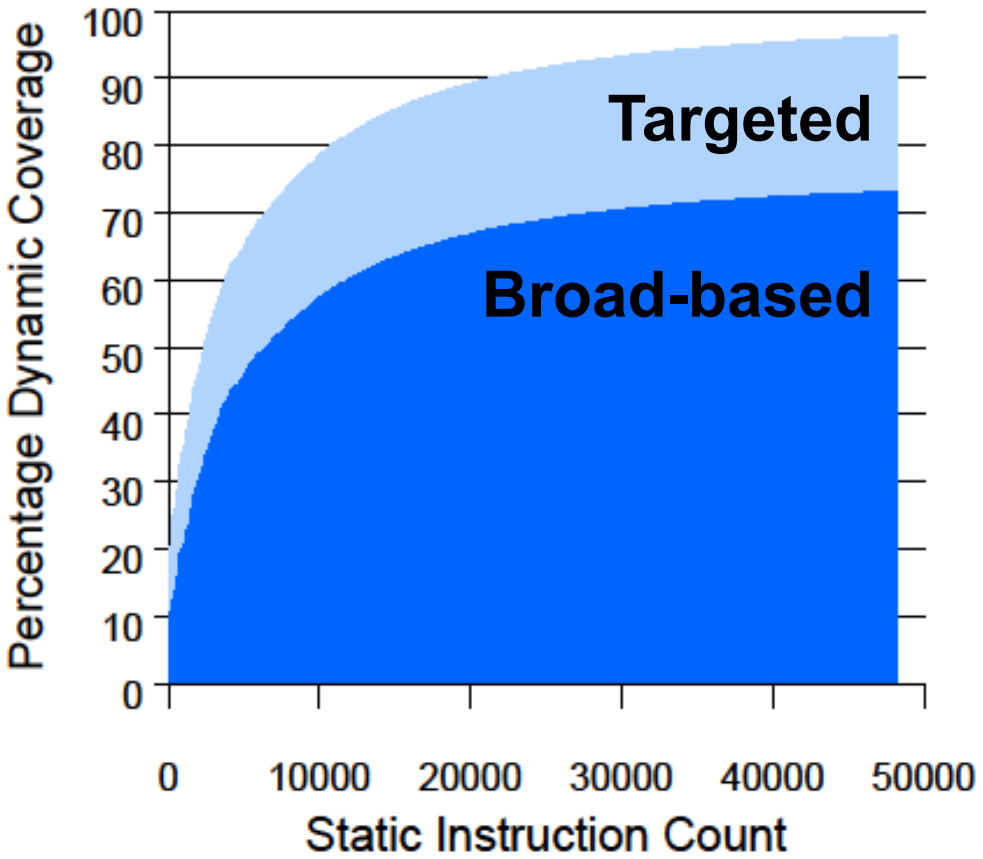
Applying C-cores to Android

- Android is well-suited for c-cores
  - Core set of commonly used applications
  - Libraries are hot code
  - Dalvik virtual machine is hot code
  - Libraries, Dalvik, and kernel & application hotspots → c-cores
  - Relatively short hardware replacement cycle



Android Workload Profile

- Profiled common Android apps to find the hot spots, including:
  - Google: Browser, Gallery, Mail, Maps, Music, Video
  - Pandora
  - Photoshop Mobile
  - Robo Defense game
- Broad-based c-cores
  - 72% code sharing
- Targeted c-cores
  - 95% coverage with just 43,000 static instructions (approx. 7 mm^2)

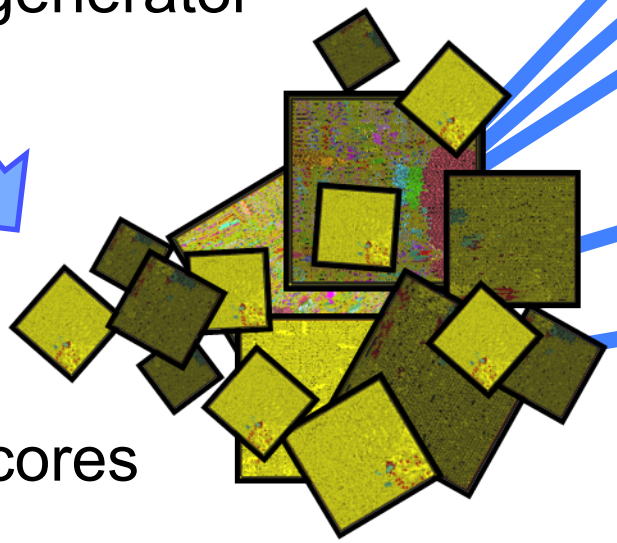


GreenDroid: Using c-cores to reduce energy in mobile application processors

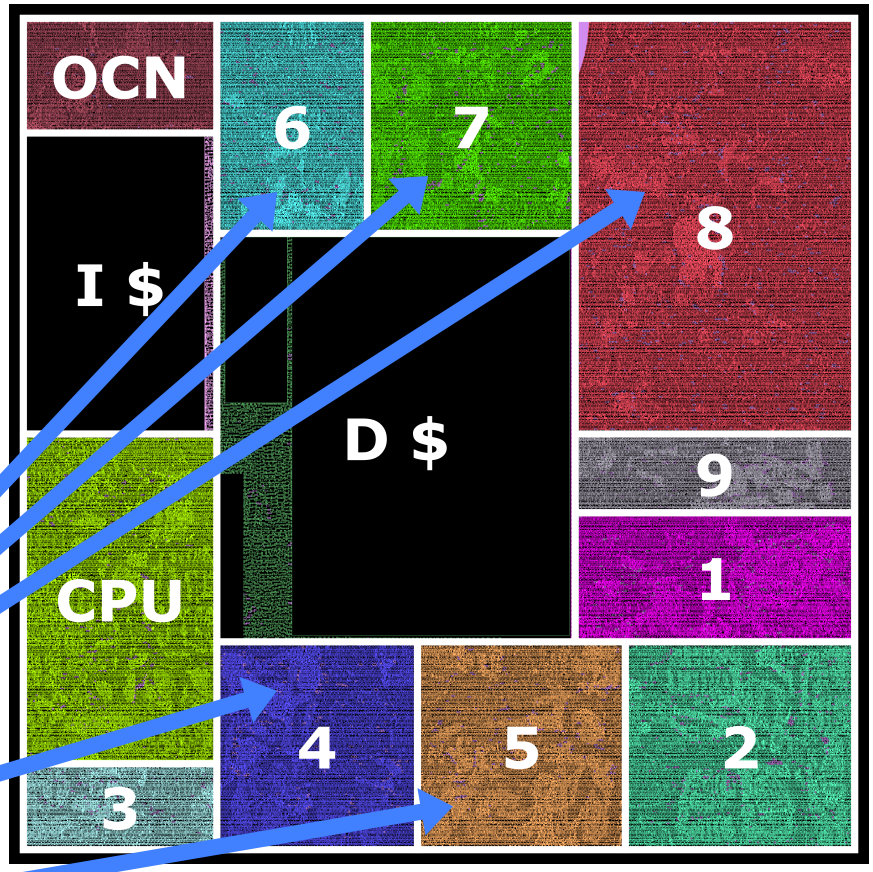
"The GreenDroid Mobile Application Processor: An Architecture for Silicon's Dark Future," Goulding-Hotta et al., IEEE Micro Mar./Apr. 2011

Android workload

Automatic c-core generator



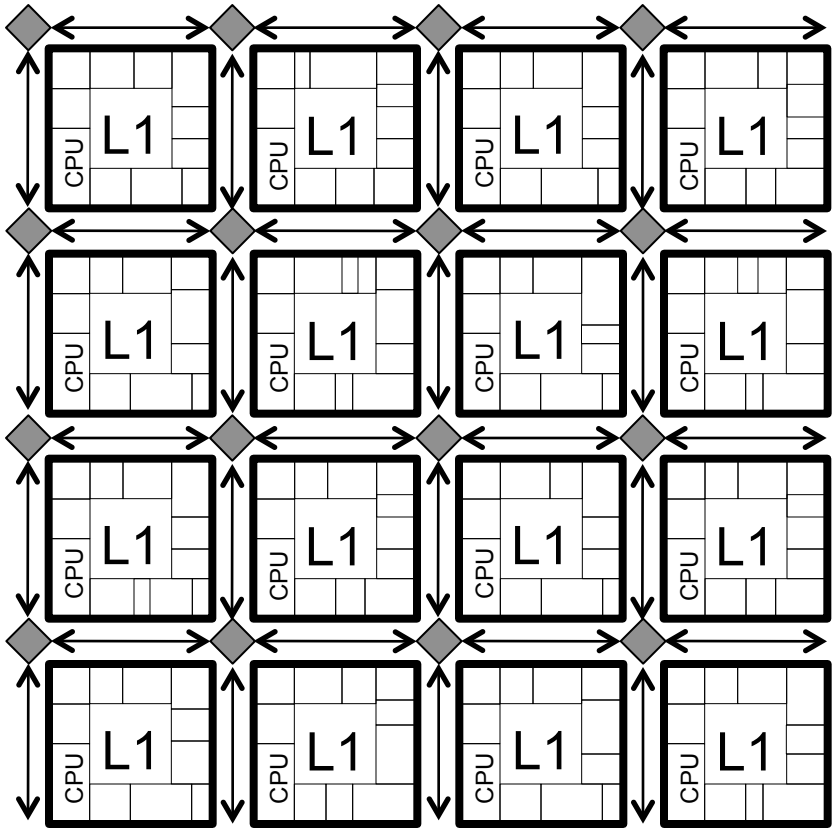
C-cores



Placed-and-routed chip with 9 Android c-cores

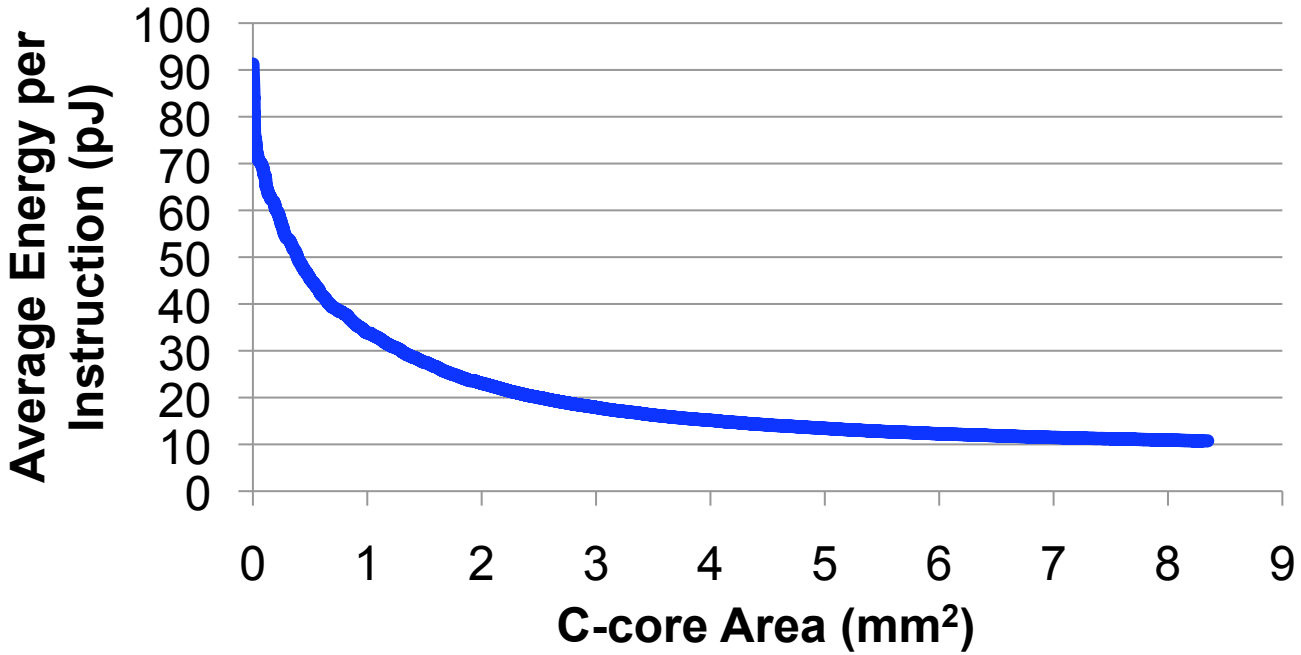
GreenDroid Tiled Architecture

- Tiled lattice of 16 cores
  (arch. scalability)
- Each tile contains
  - 6-10 Android c-cores (~125 total)
  - 32 KB D-cache (shared with CPU)
  - MIPS processor
    - 32 bit, in-order, 7-stage pipeline
    - 16 KB I-cache
    - Single-precision FPU
  - On-chip network router



GreenDroid: Energy per Instruction

- More area dedicated to c-cores yields higher execution coverage and lower energy per instruction (EPI)



- 7 mm^2 of c-cores provides:
  - 95% execution coverage
  - 8x energy savings over MIPS core

What kinds of hotspots turn into GreenDroid c-cores?

C-core	Library	# Apps	Coverage (est., %)	Area (est., mm²)	Broad-based
dvmInterpretStd	libdvm	8	10.8	0.414	Y
scanObject	libdvm	8	3.6	0.061	Y
S32A_D565_Opaque_Dither	libskia	8	2.8	0.014	Y
src_aligned	libc	8	2.3	0.005	Y
S32_opaque_D32_filter_DXDY	libskia	1	2.2	0.013	N
less_than_32_left	libc	7	1.7	0.013	Y
cached_aligned32	libc	9	1.5	0.004	Y
.plt	<many>	8	1.4	0.043	Y
memcpy	libc	8	1.2	0.003	Y
S32A_Opaque_BlitRow32	libskia	7	1.2	0.005	Y
ClampX_ClampY_filter_affine	libskia	4	1.1	0.015	Y
DiagonalInterpMC	libomx	1	1.1	0.054	N
blitRect	libskia	1	1.1	0.008	N
calc_sbr_synfilterbank_LC	libomx	1	1.1	0.034	N
inflate	libz	4	0.9	0.055	Y
. . .	. . .	. . .	. . .	. . .	. . .

GreenDroid: Projected Energy

Aggressive mobile application processor (45 nm, 1.5 GHz)	91 pJ/instr.
GreenDroid c-cores	8 pJ/instr.
GreenDroid c-cores + cold code (est.)	12 pJ/instr.

- GreenDroid c-cores use 11x less energy per instruction than an aggressive mobile application processor
- Including cold code, GreenDroid will still save ~7.5x energy

Conclusions

- The utilization wall leads to exponential worsening of the dark silicon problem and forces us to change how we build processors
- Conservation cores use dark silicon to attack the utilization wall by reducing energy across all hot code, including irregular code.
- GreenDroid will demonstrate the benefits of c-cores for mobile application processors
- We are developing GreenDroid, a 45 nm tiled prototype, at UCSD

For the details:

- Conservation Cores: Reducing the Energy of Mature Computations, *ASPLOS 2010.*
- GreenDroid: A Mobile Application Processor for a Future of Dark Silicon, *HOTCHIPS 2010.*
- Efficient Complex Operators for Irregular Codes, *HPCA 2011.*
- GreenDroid: An Architecture for Silicon's Dark Future, *IEEE Micro 2011. (Out this month!)*

greendroid.org