GreenDroid: A Mobile Application Processor for a Future of Dark Silicon

Nathan Goulding, Jack Sampson, Ganesh Venkatesh, Saturnino Garcia, Joe Auricchio, Jonathan Babb⁺, Michael B. Taylor, and Steven Swanson

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

⁺ CSAIL, Massachusetts Institute of Technology
Where does dark silicon come from?
(And how dark is it going to be?)

Utilization Wall:

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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Device count</td>
<td>$S^2$</td>
</tr>
<tr>
<td>Device frequency</td>
<td>$S$</td>
</tr>
<tr>
<td>Device power (cap)</td>
<td>$1/S$</td>
</tr>
<tr>
<td>Device power ($V_{dd}$)</td>
<td>$1/S^2$</td>
</tr>
<tr>
<td><strong>Utilization</strong></td>
<td><strong>1</strong></td>
</tr>
</tbody>
</table>

### Leakage-limited scaling

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Device count</td>
<td>$S^2$</td>
</tr>
<tr>
<td>Device frequency</td>
<td>$S$</td>
</tr>
<tr>
<td>Device power (cap)</td>
<td>$1/S$</td>
</tr>
<tr>
<td><strong>Device power ($V_{dd}$)</strong></td>
<td>~$1$</td>
</tr>
<tr>
<td><strong>Utilization</strong></td>
<td><strong>$1/S^2$</strong></td>
</tr>
</tbody>
</table>
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
  - 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

The utilization wall will change the way everyone builds processors.
Utilization Wall: Dark Implications for Multicore

Spectrum of tradeoffs between # of cores and frequency

Example:
65 nm → 32 nm (S = 2)

4 cores @ 1.8 GHz

2x4 cores @ 1.8 GHz
(8 cores dark, 8 dim)

(Industry’s Choice)

4 cores @ 2x1.8 GHz
(12 cores dark)
What do we do with dark silicon?

**Goal:** Leverage dark silicon to scale the utilization wall

**Insights:**
- Power is now more expensive than area
- Specialized logic can improve energy efficiency (10–1000x)

**Our approach:**
- Fill dark silicon with specialized cores to save energy on common applications
- Provide focused reconfigurability to handle evolving workloads
Conservation Cores
"Conservation Cores: Reducing the Energy of Mature Computations," Venkatesh et al.,
ASPLOS '10

- Specialized circuits for reducing energy
  - Automatically generated from hot regions of program source code
  - Patching support future-proofs the hardware

- Fully-automated toolchain
  - Drop-in replacements for code
  - Hot code implemented by c-cores, cold code runs on host CPU
  - HW generation/SW integration

- Energy-efficient
  - Up to 18x for targeted hot code
The C-core Life Cycle

(a) Stable applications

(b) Patchable c-core specifications

(c) Conservation cores

(d) Many-core processor with c-cores

(e) Patching-Aware Compiler

Versions released over time

Extracted energy-intensive code regions

1.9 1.2 1.21 1.22 1.3
2.96 3.4 3.5 4.2 4.21

12
Outline

- Utilization wall and dark silicon
- GreenDroid
- Conservation cores
- GreenDroid energy savings
- Conclusions
Emerging Trends

The *utilization wall* is exponentially worsening the dark silicon problem.

Specialized architectures are receiving more and more attention because of energy efficiency.

*Mobile application processors* are becoming a dominant computing platform for end users.
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 power constraints
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 mm²)
GreenDroid: Applying Massive Specialization to Mobile Application Processors

Android workload

Automatic c-core generator

Conservation cores (c-cores)

Low-power tiled multicore lattice
GreenDroid Tiled Architecture

- Tiled lattice of 16 cores
- 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 Tile Floorplan

- 1.0 mm² per tile
- 50% C-cores
- 25% D-cache
- 25% MIPS core, I-cache, and on-chip network
GreenDroid Tile Skeleton

- 45 nm process
- 1.5 GHz
- ~30k instances

- Blank space is filled with a collection of c-cores
- Each tile contains different c-cores
Outline

- Utilization wall and dark silicon
- GreenDroid
- Conservation cores
- GreenDroid energy savings
- Conclusions
Constructing a C-core

- C-cores start with source code
  - Can be irregular, integer programs
  - Parallelism-agnostic

- Supports almost 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

```c
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 → Place & Route
  - 45 nm process
  - Synopsys CAD flow
    - Synthesis
    - Placement
    - Clock tree generation
    - Routing

0.01 mm², 1.4 GHz
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 mm²
  - Frequencies from 1.0 to 1.4 GHz

<table>
<thead>
<tr>
<th>Application</th>
<th># C-cores</th>
<th>Area (mm²)</th>
<th>Frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bzip2</td>
<td>1</td>
<td>0.18</td>
<td>1235</td>
</tr>
<tr>
<td>cjpeeg</td>
<td>3</td>
<td>0.18</td>
<td>1451</td>
</tr>
<tr>
<td>djpeeg</td>
<td>3</td>
<td>0.21</td>
<td>1460</td>
</tr>
<tr>
<td>mcf</td>
<td>3</td>
<td>0.17</td>
<td>1407</td>
</tr>
<tr>
<td>radix</td>
<td>1</td>
<td>0.10</td>
<td>1364</td>
</tr>
<tr>
<td>sat solver</td>
<td>2</td>
<td>0.20</td>
<td>1275</td>
</tr>
<tr>
<td>twolf</td>
<td>6</td>
<td>0.25</td>
<td>1426</td>
</tr>
<tr>
<td>viterbi</td>
<td>1</td>
<td>0.12</td>
<td>1264</td>
</tr>
<tr>
<td>vpr</td>
<td>1</td>
<td>0.24</td>
<td>1074</td>
</tr>
</tbody>
</table>
C-core Energy Efficiency: Non-cache Operations

- Up to 18x more energy-efficient (13.7x on average), compared to running on the MIPS processor
Where do the energy savings come from?

- D-cache: 6%
- I-cache: 23%
- Fetch/Decode: 19%
- Reg. File: 14%
- Datapath: 38%

MIPS baseline: 91 pJ/instr.

- D-cache: 6%
- Datapath: 3%

C-cores: 8 pJ/instr.

Energy Saved: 91%
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
Outline

- Utilization wall and dark silicon
- GreenDroid
- Conservation cores
- GreenDroid energy savings
- Conclusions
GreenDroid: Energy per Instruction

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

- 7 mm² of c-cores provides:
  - 95% execution coverage
  - 8x energy savings over MIPS core
What kinds of hotspots turn into GreenDroid c-cores?

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

...
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
Project Status

Completed

- Automatic generation of c-cores from source code to place & route
- Cycle- and energy-accurate simulation (post place & route)
- Tiled lattice, placed and routed
- FPGA emulation of c-cores and tiled lattice

Ongoing work

- Finish full system Android emulation for more accurate workload modeling
- Finalize c-core selection based on full system Android workload model
- Timing closure and tapeout
GreenDroid Conclusions

- The utilization wall forces us to change how we build hardware

- Conservation cores use dark silicon to attack the utilization wall

- GreenDroid will demonstrate the benefits of c-cores for mobile application processors

- We are developing a 45 nm tiled prototype at UCSD
GreenDroid: A Mobile Application Processor for a Future of Dark Silicon

Nathan Goulding, Jack Sampson, Ganesh Venkatesh, Saturnino Garcia, Joe Auricchio, Jonathan Babb⁺, Michael B. Taylor, and Steven Swanson

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

⁺ CSAIL, Massachusetts Institute of Technology

Hot Chips 22 Aug. 23, 2010
Backup Slides
Automated Measurement Methodology

- C-core toolchain
  - Specification generator
  - Verilog generator

- Synopsys CAD flow
  - Design Compiler
  - IC Compiler
  - 45 nm library

- Simulation
  - Validated cycle-accurate c-core modules
  - Post-route gate-level simulation

- Power measurement
  - VCS + PrimeTime