Conservation Cores: Reducing the Energy of Mature Computations

Ganesh Venkatesh, Jack Sampson, Nathan Goulding, Saturnino Garcia, Vladyslav Bryksin, Jose Lugo-Martinez, Steven Swanson, Michael Bedford Taylor

Department of Computer Science and Engineering, University of California, San Diego
The Utilization Wall

- **Scaling theory**
  - Transistor and power budgets no longer balanced
  - Exponentially increasing problem!

- **Experimental results**
  - Replicated 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>Utilization</td>
<td>$1$</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>Device power ($V_{dd}$)</td>
<td>$\sim 1$</td>
</tr>
<tr>
<td>Utilization</td>
<td>$1/S^2$</td>
</tr>
</tbody>
</table>
The Utilization Wall

- Scaling theory
  - Transistor and power budgets no longer balanced
  - Exponentially increasing problem!

- Experimental results
  - Replicated small datapath
  - More ‘Dark Silicon’ than active

- Observations in the wild
  - Flat frequency curve
  - “Turbo Mode”
  - Increasing cache/processor ratio

![Graph showing expected utilization for fixed area and power budget]
The Utilization Wall

- **Scaling theory**
  - Transistor and power budgets no longer balanced
  - Exponentially increasing problem!
- **Experimental results**
  - Replicated small datapath
  - More ‘Dark Silicon’ than active
- **Observations in the wild**
  - Flat frequency curve
  - “Turbo Mode”
  - Increasing cache/processor ratio

![Diagram showing Utilization @ 300mm² & 80w

- 90nm TSMC: 17.6%
- 45nm TSMC: 6.5%
- 32nm ITRS: 3.3%

- 3x
- 2x
- 3.3%
The Utilization Wall

- Scaling theory
  - Transistor and power budgets no longer balanced
  - Exponentially increasing problem!

- Experimental results
  - Replicated small datapath
  - More ‘Dark Silicon’ than active

- Observations in the wild
  - Flat frequency curve
  - “Turbo Mode”
  - Increasing cache/processor ratio
The Utilization Wall

- Scaling theory
  - Transistor and power budgets no longer balanced
  - Exponentially increasing problem!

- Experimental results
  - Replicated small datapath
  - More ‘Dark Silicon’ than active

- Observations in the wild
  - Flat frequency curve
  - “Turbo Mode”
  - Increasing cache/processor ratio

- We’re already here
Utilization Wall: Dark Implications for Multicore

Spectrum of tradeoffs between # cores and frequency.

e.g.; take
65 nm $\rightarrow$ 32 nm;
i.e. ($s = 2$)

4 cores @ 3 GHz

2x4 cores @ 3 GHz
(8 cores dark)
(Industry’s Choice)

4 cores @ 2x3 GHz
(12 cores dark)
What do we do with Dark Silicon?

- **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 cores to save energy on common apps
  - Power savings can be applied to other program, increasing throughput

- **C-cores provide an architectural way to trade area for an effective increase in power budget!**
Conservation Cores

- Specialized cores for reducing energy
  - Automatically generated from hot regions of program source
  - Patching support future proofs HW

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

- Energy efficient
  - Up to 16x for targeted hot code
The C-Core life cycle

Stable Applications

(a)

Extracted Energy-intensive Code Regions

(b)

Patching-Aware Compiler

(e)

Many-core Processor with C-cores

(d)

Conservation Cores

(c)

Version Released Over Time

1.3

4.21

4.2

1.22

1.21

3.5

3.4

2.96

1.2

0.9
Outline

- The Utilization Wall
- Conservation Core Architecture & Synthesis
- Patchable Hardware
- Results
- Conclusions
Constructing a C-Core

- C-Cores start with source code
  - Parallelism agnostic

- C code supported
  - Arbitrary memory access patterns
  - Complex control flow
  - Same cache memory model as processor
  - Function call interface

```c
sumArray(int n, int *a) {
    int i = 0;
    int sum = 0;
    for(; i<n; i++)
    {
        sum += a[i];
    }
    return(sum);
}
```
Constructing a C-Core

- Compilation
  - C-Core isolation
  - SSA, infinite register, 3-address
  - Direct mapping from CFG, DFG
  - Scan chain insertion

- Verilog to Place & Route
  - TSMC 45nm libraries
  - Synopsys CAD flow
    - Synthesis
    - Placement
    - Clock Tree Generation
    - Routing
C-Core for sumArray

Gold – Control path
Blue – Registers
Green – Data path

Post-route Std. Cell layout of an actual C-Core generated by our toolchain

0.01 mm$^2$, 1.4 GHz
A C-Core enhanced system

- Tiled multiprocessor environment
  - Homogeneous interfaces, heterogeneous resources

- Several C-Cores per tile
  - Different types of C-cores on different tiles

- Each C-Core interfaces with 8-stage MIPS core
  - Scan chains, cache as interfaces
Outline

- The Utilization Wall
- Conservation Core Architecture & Synthesis
- Patchable Hardware
- Results
- Conclusions
Patchable Hardware

- Future versions of hot code regions may have changes
  - Need to keep HW usable
  - C-Cores unaffected by changes to cold regions

- General exception mechanism
  - Trap to SW
  - Can support any changes
Reducing the cost of change

- Examined versions of applications as they evolved
  - Many changes are straightforward to support

- Simple lightweight configurability
  - Preserve structure
  - Support only those changes commonly seen

<table>
<thead>
<tr>
<th>Structure</th>
<th>Replaced by</th>
</tr>
</thead>
<tbody>
<tr>
<td>adder</td>
<td>AddSub</td>
</tr>
<tr>
<td>subtractor</td>
<td></td>
</tr>
<tr>
<td>comparator(GE)</td>
<td>Compare6</td>
</tr>
<tr>
<td>bitwise AND, OR, XOR</td>
<td>BitwiseALU</td>
</tr>
<tr>
<td>constant value</td>
<td>32-bit register</td>
</tr>
</tbody>
</table>
Patchability overheads

- **Area overhead**
  - Split between generalized datapath elements and constant registers

- **Power overhead**
  - 10-15% for generalized datapath elements

- **Opportunity costs**
  - Reduced partial evaluation
  - Can be large for multipliers, shifters

<table>
<thead>
<tr>
<th>Structure</th>
<th>Area ($\mu$m$^2$)</th>
<th>Replaced by</th>
<th>Area ($\mu$m$^2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>adder/subtractor</td>
<td>270/270</td>
<td>Add/Sub</td>
<td>365</td>
</tr>
<tr>
<td>comparator (GE)</td>
<td>133</td>
<td>Compare6</td>
<td>216</td>
</tr>
<tr>
<td>bitwise AND, OR</td>
<td>34/56</td>
<td>Bitwise</td>
<td>191</td>
</tr>
<tr>
<td>bitwise XOR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>constant value</td>
<td>$\sim$ 0</td>
<td>32-bit register</td>
<td>160</td>
</tr>
</tbody>
</table>
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

- The Utilization Wall
- Conservation Core Architecture & Synthesis
- Patchable Hardware
- Results
- Conclusions
Automated measurement methodology

- C-Core toolchain
  - Specification generator
  - Verilog generator

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

- Simulation
  - Validated cycle-accurate C-Core modules
  - Post-route netlist simulation

- Power measurement
  - VCS+PrimeTime

Diagram:

- Source
  - Hotspot analyzer
  - Cold code
  - Hot Code
  - Rewriter
  - gcc
  - Simulation
  - Synopsys flow
  - Power measurement
Our cadre of C-Cores

- We built 23 C-Cores for assorted versions of 5 applications
  - Both patchable and non-patchable versions of each
  - Varied in size from 0.015 to 0.326 mm²
  - Frequencies from 0.9 to 1.9GHz
C-Core hot-code energy efficiency

- Up to 16x as efficient as general purpose in-order core, 9.5x on average
System energy efficiency

- C-Cores very efficient for targeted hot code
- Amdahl’s Law limits total system efficiency
C-Core system efficiency with current toolchain

<table>
<thead>
<tr>
<th>Software</th>
<th>Patchable</th>
</tr>
</thead>
<tbody>
<tr>
<td>jpeg A</td>
<td>jpeg B</td>
</tr>
<tr>
<td>mcf A</td>
<td>mcf B</td>
</tr>
<tr>
<td>vpr A</td>
<td>vpr B</td>
</tr>
<tr>
<td>cjjpeg A</td>
<td>cjjpeg B</td>
</tr>
<tr>
<td>bzip2 A</td>
<td>Avg. A-F</td>
</tr>
</tbody>
</table>

- Base
  - Avg 33% EDP improvement
Tuning system efficiency

- Improving our toolchain’s coverage of hot code regions
  - Good news: Small numbers of static instructions account for most of execution

- System rebalancing for cold-code execution
  - Improve performance/leakage trade-offs for host core
C-Core system efficiency with toolchain improvements

With improved coverage, system components:
- Avg 61% EDP savings
- Avg 53% EDP improvement
- Avg 14% increased execution time
Conclusions

- The Utilization Wall will change how we build hardware
  - Hardware specialization increasingly promising

- *Conservation Cores* are a promising way to attack the Utilization Wall
  - Automatically generated patchable hardware
  - For hot code regions: 3.4 – 16x energy efficiency
  - With tuning: 61% application EDP savings across system
  - 45nm tiled C-Core prototype under development @ UCSD

- Patchability allows C-Cores to last for ten years
  - Lasts the expected lifetime of a typical chip