Dark Silicon and its Implications for Future Processor Design

Max Menges
E-Mail: mmenges@mathphys.fsk.uni-heidelberg.de

***Abstract*—Energy efficiency has always been a goal when designing a chip or system to reduce the costs of power consumption and cooling. With the breakdown of Dennard scaling and chip designs running up the utilization wall, designing an energy efficient system will directly benefit performance as more parts of a chip can be turned on at the same time.**

As a result of this, multicore CPUs have been introduced and hardware accelerators like GPUs have become increasingly important in HPC applications. While suitable for regular floating point arithmetics, manycore designs would perform badly as a general purpose CPU.

As a response to this problem, energy efficient, highly specialized cores targeting specific, frequently used functions in an application are introduced. These cores are called conservation cores. Simulations show that augmenting a design with conservation cores can improve energy efficiency by 23% without loss of performance.

I. Introduction

Moore's law predicts the doubling in transistor count per chip every two years. To achieve this, transistor geometries have been continually sized down to accommodate the extra devices on the chip. Now as feature sizes shrink with each new manufacturing technology, leakage current becomes one of the dominant sources of power consumption. The energy consumption no longer scales at the same rate as geometries shrink and thus the overall consumption in a fixed chip area rises. Therefore, the number of devices which can actively switch at full speed within the chips power budget will decrease. This is called the *utilization wall*. The remaining silicon which is left unpowered is referred to as *dark silicon* and will grow exponentially with each new technology generation. As a result, designing energy efficient chips will not only reduce the operating cost but directly improve performance.

Symptomatic for this development is Intel's Nehalem architecture [1] which allows a core to run at a higher frequency at the cost of switching off other cores. ARM follows the same idea with the big.LITTLE architecture [2] which uses heterogeneous cores, one being 'just powerful enough' for small tasks while the other, larger, more powerful core is only turned on if the extra processing power is needed. Another approach are so called *conservation cores* which implement software functions directly in hardware structures to achieve significant energy savings per instruction.

Section II discusses classical scaling theory and the origin of dark silicon as a result of post-Dennard scaling. As a result of this, multi- and manycore designs were introduced to the market. How this helps improve performance in post-Dennard scaling is explained in section III. Finally another approach to the dark silicon problem is introduced by using conservation cores. Section IV describes the GreenDroid architecture and analyzes the gains in energy efficiency that can be made by using conservation cores.

II. Dark Silicon

A. Classical Scaling

In the past, significant performance gains could be obtained with each new process generation. Frequencies were scaled up and geometries shrunk, resulting in faster and more transistors per chip while not increasing the overall power budget. This was possible due to the so called *Dennard scaling*, which states that as geometries get smaller, the energy density in a transistor stays constant since voltage and current in the transistor scale downward as well [3].

The classical scaling column in table I shows the principles of technology scaling before the breakdown of Dennard scaling.

The transistor dimensions W and L are scaled by a factor of $\frac{1}{S}$, where usually $S = 1.4$ resulting in a technology shrink from e.g. $45nm$ to $32nm$. This will result in an approximately doubled device count D_S of the scaled technology in a constant area.

$$D_S = \frac{A}{W_S L_S} = \frac{A}{\frac{1}{S^2} W L} = S^2 \cdot D \qquad (1)$$

At the same time, the supply voltage V_{dd} and oxide thickness t_{ox} are scaled by a factor of $\frac{1}{S}$ as well which results in a reduction of gate capacitances of $\frac{1}{S}$.

The dynamic switching power, which in this case is the main source of power consumption, comes from

Parameter	Description	Relation	Classical Scaling	Leakage Limited
B	Power budget		1	1
A	Area		1	1
V_{dd}, V_{th}	Supply & threshold voltages		$\mathbf{1/S}$	$\mathbf{1}$
t_{ox}	Oxide thickness		$1/S$	$1/S$
W, L	Transistor dimensions		$1/S$	$1/S$
I_{sat}	Saturation current	WV_{dd}/t_{ox}	$\mathbf{1/S}$	$\mathbf{1}$
p	Power per device	CV^2f	$\mathbf{1/S^2}$	$\mathbf{1}$
C	Gate capacitance	WL/t_{ox}	$1/S$	$1/S$
f	frequency	$\frac{I_{sat}}{CV_{dd}}$	S	S
D	Devices per chip	A/WL	S^2	S^2
P	Full die, full power	Dp	$\mathbf{1}$	$\mathbf{1/S^2}$
U	Utilization	B/P	$\mathbf{1}$	$\mathbf{1/S^2}$

Table I
CLASSICAL AND LEAKAGE LIMITED SCALING

loading and unloading the gate (load) capacitances C_L driven by a transistor.

$$P_{dyn} = \alpha C_L V^2 f \tag{2}$$

The switching probability α states how often the transistor switches. Even though the frequency f increases by a factor S, the dynamic power per transistor is reduced by $\frac{1}{S^2}$ which counteracts the growing transistor count and therefore the overall power consumption of a scaled down chip stays constant.

The increased frequency directly impacts the performance of the chip. The now available transistors can be used to further improve the performance. More functional units can be added if this is the bottleneck of the design and application. Another common bottleneck of most chips is the access to the main memory. On chip caches can be added or extended to hide this memory access latency. Along with data caches, instruction caches, sophisticated branch prediction and preemptive fetch and execute modules can be implemented.

All these structures contain large amounts of parallel switching transistors which consume a lot of energy. This is not a problem when energy consumption scales proportional to geometries, but with the breakdown of Dennard scaling, these structures contribute to the dark silicon phenomenon.

B. Breakdown of Dennard Scaling

Up until around 2005, or the 65 nm process, this method of technology scaling and design paradigm proved feasible. At this point certain physical and processing limits were reached.

Classical scaling relied on supply and threshold voltages scaling accordingly. With new process generations, the supply voltage could no longer be lowered as needed. One reason for this is the increasing subthreshold leakage current which is exponentially proportional to the threshold voltage V_{th}:

$$P_{leak} \propto e^{\frac{V_{GS}-V_{th}}{nV_T}} \tag{3}$$

Subthreshold leakage current is a current flowing from the transistors drain to source terminal when the transistor is nominally turned off, i.e. the gate voltage is below the transistor threshold voltage. Unlike the dynamic switching power, this contribution to the overall power consumption cannot be eliminated by simply not using the transistor but instead the supply voltage V_{dd} would have to be disconnected from the device. This is not feasible for caches etc. as they still have to hold the data values even though the transistors are not switching. Studies have shown, that subthreshold leakage currents can make up about 50% of a chips total power consumption at the $90nm$ node [4].

Another source of increased energy consumption per transistor are quantum mechanical tunneling effects at the thin gate oxide. With an increasingly thin gate oxide thickness of only a few nm or a few layers of silicon, electrons may tunnel through the gate insulator into the channel further adding to the power consumption of the transistor.

These limitations lead to the supply voltage V_{dd} not scaling by a factor of $\frac{1}{S}$ but staying constant. This leads to a scaled dynamic power consumption per device of

$$P_{dynS} = \alpha \cdot \frac{1}{S} C_L V^2 S f \tag{4}$$
$$= \alpha \cdot C_L V^2 f \tag{5}$$
$$= P_{dyn} \tag{6}$$

which is the same as the consumption of the previous technology node. With the increased device count

$$D_S = \frac{A}{W_S L_S} \tag{7}$$
$$= \frac{A}{\frac{1}{S^2} W L} \tag{8}$$
$$= S^2 \cdot D \tag{9}$$

the power of the full chip at full frequency will increase by a factor of S^2.

Assuming a constant power budget B, the utilization $U = B/P$, i.e. the percentage of the chip which can be turned on within the power budget, will drop by $\frac{1}{S^2}$ with each new process generation.

The scaling factors of post Dennard scaling are shown in the leakage limited column of table I.

Table II shows the severity of this problem. The authors of [5] have implemented a 64-bit adder design in $90nm$ and $45nm$ TSMC processes in a fixed area of $300mm^2$. The available chip area was then covered with the 64-bit in- and output registered adders. The energy consumption of each chip at full frequency was measured and from this the utilization at $80W$ deducted. From the results of the silicon implementations and data provided by the ITRS, projections for the $32nm$ node have been made.

The results show, that at $90nm$ the chip would need $455W$. When operated within the $80W$ budget, only 17.6% of the chip can actively be used. At $45nm$ the needed energy for the whole chip rises to $1225W$ resulting in only 6.5% utilization. Since the scaling factor from $90nm$ to $45nm$ is $S = \frac{1}{2}$, the expected reduction in utilization would be $\frac{1}{4}$. The actual reduction is by a factor of about $\frac{1}{3}$. This is attributed to improvements made to the process and standard cells between the two processes.

Interpolating from these results, the energy needed to power a $300mm^2$ chip in a $32nm$ process at full frequency would be $2401W$ which would mean a utilization of only 3.3% and even less at future nodes.

III. From Single to Multi- and Manycore Designs

As stated above, the breakdown of Dennard scaling happened around 2005. But since then geometries have continued to be scaled down and chip performance has been increased. For this, new design paradigms have been introduced. Concurrent to the breakdown of Dennard scaling, multicore CPUs have been introduced to the market and have since become the standard in desktop computers. In the high performance computing sector, an emphasis has been put on energy efficient accelerators like GPUs and FPGAs as an alternate way to increase computing power.

The idea behind this development is the reduction of clock speeds. Even though the native switching speeds of transistors have continued to increase with each technology node, processor clocking frequencies have remained largely constant in the last years.

When reducing the switching frequency of a transistor, the supply voltage can also be lowered since a lower saturation current is needed to charge the load capacitors. Therefore the dynamic switching power is roughly proportional to f^3. A reduction of clock speed to 80% will result in approximately 50% dynamic power consumption. Assuming the performance of a single core CPU is directly proportional to its operating frequency, this would mean a loss of 20% performance. Duplicating the core design and adding a second identical core to the die will result in the original power consumption but will increase performance by a factor of 1.6. This explains why processor clocking speeds have been largely stable in the last few years.

Throughput orientated accelerators like GPUs have taken this principle one step further. Instead of a few cores, a few thousand cores are implemented on a die and the clock frequency is lowered even further. Typical GPU speed being at $700MHz$ to $800MHz$. These accelerators are not optimized for latency but for throughput and do not rely on high switching frequencies. Their strength lies in handling large numbers of regular floating point operations. This makes the manycore approach rather useless for the mostly irregular task of a general purpose CPU.

ARM has introduced the big.LITTLE architecture which combines two different cores on one die to improve energy efficiency. The idea behind this is, that for most tasks it is sufficient to run them on the LITTLE smaller, slower but more energy efficient core. If required the task can be scheduled to the big core which is faster but also consumes more energy. Ideally a core is just about fast and powerful enough to complete a given task in reasonable time as not to waste energy when over-utilizing a core.

Process	90 nm TSMC	45 nm TSMC	32 nm ITRS
Frequency [GHz]	2.1	5.2	7.3
Full Chip Watts	455	1225	2401
Utilization at 80 W	17.6%	6.5%	3.3%

Table II
EXPERIMENTAL RESULTS FOR THE UTILIZATION WALL

The proposed GreenDroid architecture follows the same approach of further specifying the hardware to the software but in this case a core is not a general purpose processing unit but rather specialized hardware targeted at a specific software function with the sole goal of reducing energy per instruction.

IV. CONSERVATION CORES

The GreenDroid is a chip architecture designed for mobile phones running the Android operating system. It utilizes *conservation cores* or *c-cores* to reduce energy consumption per instruction. A c-core implements a software function directly in hardware, thus eliminating the need for instruction fetch and decode mechanisms as well as using a highly specialized data path.

A mobile processor and the Android software stack are ideal for this approach for various reasons:

- The Android software stack has a large stable and well tested code base. Major changes to the code are unlikely and only minor patches to the code are introduced.
- User applications run in a virtual machine called Dalvik, therefore no new user code has to be anticipated. It is sufficient to implement the functions of the VM as c-cores.
- Even though many applications are available for download, Android has a few commonly used applications such as a web browser, e-mail client, media player etc.
- Smartphone chips have a tight power budget of about $3W$.
- The usually short replacement cycles of approximately two years for a smartphone ensure that the c-core will not be outdated.

A. The GreenDroid Architecture

The GreenDroid consists of 16 tiles connected to each other via a point-to-point mesh interconnect as an on chip network (see figure 1).

Each tile uses a standard 32-bit in-order MIPS core and contains instruction and data caches for the MIPS core. Additionally the tile contains an array of 8 to 15 c-cores targeting specific software functions (see figure 2). Each tile is therefore unique.

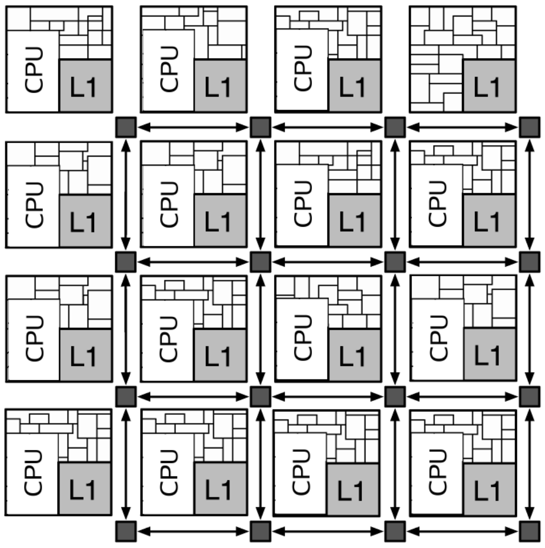

Figure 1. Layout of the GreenDroid Chip Architecture, source: [6]

The MIPS core is coupled to the c-cores via the L1 cache in the tile. Communication with the c-cores and starting an execution on a c-core is done via the scan chain interface.

C-cores targeting parts of the same software function or those for which the use correlates, are clustered together in the same tile to improve performance.

B. Generating Cores

To generate the c-cores for an application, the workload of the application has to be characterized. For this, the regions of ‘hot’ code, i.e. most frequently used parts of the code, have to be identified. These are sections of the code which benefit the most from the use of c-cores in terms of energy efficiency.

Once the part of the code which is to be executed on a c-core are identified, the code is translated to verilog code and synthesized. This is done by translating the control flow graph (CFG) of the code into a finite state machine (FSM) of the c-core. The CFG is now closely coupled to the FSM of the c-core which makes later

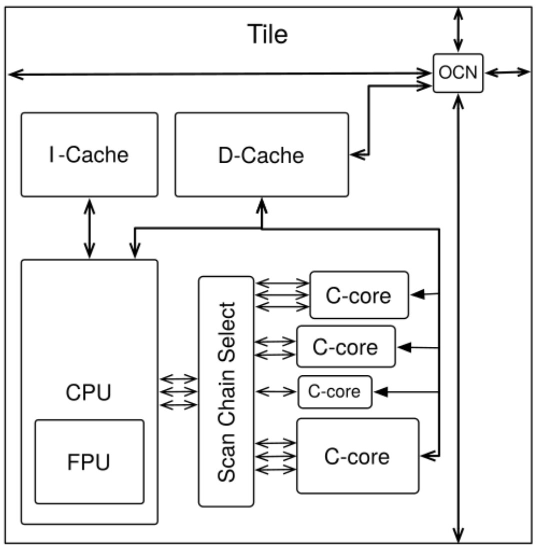

Figure 2. Schematic of One Tile in the GreenDroid Chip, source: [6]

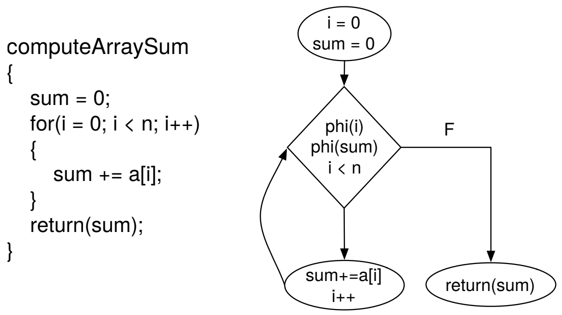

Figure 3. The Control Flow Graph of the Code Will Determine the State Machine of the C-Core, source: [5]

changes easier. An example data path and FSM are shown in figure 4.

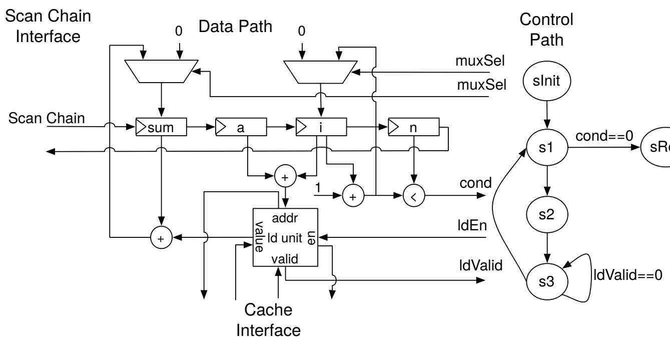

Figure 4. Data Path and FSM of a C-Core, source [5]

Afterwards scan chains are inserted to connect all registers of the core. Apart from the L1 cache, the CPU can communicate with the c-cores only through these scan chains. Function arguments can be passed through the scan chain interface to the c-cores and the internal state of the datapath can be read back to the CPU. Once all parameters and configuration bits are passed to the c-core, the execution can be started by the CPU using a single bit master scan chain. When the execution is complete, the c-core raises an exception and passed control back to the CPU.

Once all c-cores are in place, the code can be compiled for the c-core enabled design. The compiler has to be aware of the c-core functionality. If it encounters a portion of the code that can be executed on a c-core, it will insert a function stub which allows execution either on the c-core or on the host CPU. In this way, the CPU can determine at runtime if an appropriate c-core is available and if not, execute the code itself.

C. *Patching Cores*

To increase the useful lifespan of a c-core enabled design, the c-cores are fitted with patching mechanisms which allow a core to continue working after minor changes have been made to the code. Three most common categories of code changes have been identified:

- Hard coded constants, e.g. loop boundaries, initial values etc., might change from one version to another.
- Operators might be replaced, e.g. a plus becoming a minus or comparison operators are reversed.
- The structure of the CFG might change, code blocks moved or entire sections of code are deleted or replaced.

For each of these categories a mechanism is implemented which allows the c-core to adapt to the new code.

To enable the change of constants, rather than hard coding them in the c-core, all constants are replaced with a register. The value of the register can then be set at runtime using the scan chain interface.

Another common change is the altering of an operator. Therefore the initial operators are replaced with generalized version and a control bit selecting the correct operator. E.g. an adder is replaced by an adder-subtracter and comparison operators by a generalized comparison operator.

Lastly, a new version might significantly change a portion of the code. This would render the corresponding c-core useless even if a large part of the code remains unchanged. To account for this, each edge of the state machine is augmented with an exception bit. If the c-core encounters a part of the code which for any reason cannot be executed on the core, it raises an exception and control is transfered back to the host CPU. The CPU can then readout the internal state of all registers at the time of the exception and continue running the code on its own until a point is reached where the c-core can take over again. The values of all registers are updated using the scan chain and control is again transfered back to the c-core. In this way, control can bounce back and forth between CPU and c-core until the function call is complete.

All these changes are implemented before synthesizing the code to hardware. Of course, adding more (and in some cases not used) functionality to the design takes up more space on the chip but ensures a longer life cycle of the c-core. The core can then in turn contribute longer to the energy efficient execution of the code. This justifies the overhead originating from making the core patchable.

D. Results

For analysis of the energy savings that can be obtained by using c-cores, a complete tile of the GreenDroid architecture has been designed and simulated. The c-cores implement functions from various versions of bzip2, cjpeg, djpeg, mcf[1] and vpr[2]. Together these span about 3600 lines of C code and the time each program spends in these function ranges from 1.3% to 71.1% of total execution time.

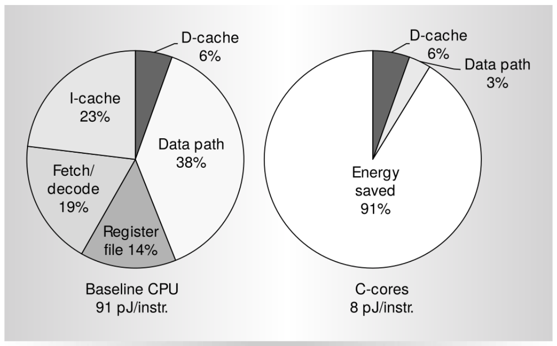

Figure 5. Energy Per Instruction of the Baseline CPU Compared to a C-Core, source [6]

Figure 5 shows the energy consumption per instruction from the baseline MIPS CPU compared to the execution on a c-core. The $91pJ$ per instruction can be reduced by 91% to just $8pJ$ per instruction on the c-core. Most of the energy savings come from the c-core not needing any instruction cache or fetch and decode mechanisms which are by design hard wired into the core. The core also has no need of a traditional register file as data is directly inserted to the core via the scan chain. The last part of the energy savings come from the highly specialized data path.

[1]A memory intensive integer benchmark

[2]Place and route tool for FPGAs

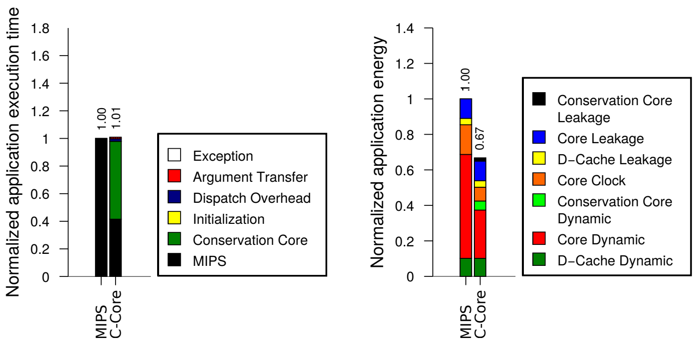

Figure 6. Detailed Power Analysis of Code Running on a General Purpose CPU Compared to a C-Core, source [5]

Figure 6 compares execution times of code run exclusively on the MIPS CPU and the same code run on the c-cores. The results are an average of the above mentioned functions and they are normalized to the baseline MIPS core.

The overall execution time increases only slightly by about 1% and about 60% of the time is spent in a c-core. By executing the code on the energy efficient c-core, the dynamic switching power of the MIPS can be drastically reduced which leads to an overall energy saving compared to the MIPS of 23%.

The results shown in figure 6 are for patchable c-cores. Without introducing the patching support, significantly higher energy reductions can be achieved and a core would require less space on the chip. For example, replacing an adder in the $45nm$ technology by an adder-subtractor would increase the area from $270\mu m^2$ to $365\mu m^2$. Identifying where patching support is not required could further increase energy efficiency and the now available space could be used to implement additional c-cores.

Another obvious goal is to increase c-core coverage. Adding support for floating point operations in c-cores would increase the amount of code which could be executed on a c-core even further. As a side effect of spending more time in the c-cores rather than the host CPU, changes to the MIPS core and peripherals could be made. Using high V_{th} transistors in the CPU would reduce leakage losses without significantly impacting performance.

V. Conclusion

Due to the dark silicon phenomenon, leakage limited scaling and traditional chip design paradigms will not yield the same performance gains as in the past. Introducing conservation cores to a design can significantly reduce its power consumption. Simulations have shown, that the power needed for a single instruction can be reduced by 91% and the overall power of a chip can be reduced by 23% and there is still a lot of room for improvement by increasing c-core coverage.

Utilizing the dark silicon in energy efficient computations can boost performance of a chip. If less power is needed per instruction, more can be run in parallel.

However, the dark silicon problem still remains and will get worse with each new technology node. Using c-cores in a design might help bridge the gap until a breakthrough in energy efficient transistors is made or a new technology replaces current CMOS logic.

References

[1] Intel, "First the tick, now the tock: Next generation intel microarchitecture (nehalem)," Intel Whitepaper, 2008.

[2] ARM, "big.little technology: The future of mobile," ARM Whitepaper, 2013.

[3] R. H. Dennard, F. H. Gaensslen, H. nien Yu, V. L. Rideout, E. Bassous, Andre, and R. Leblanc, "Design of ion-implanted mosfets with very small physical dimensions," *IEEE J. Solid-State Circuits*, p. 256, 1974.

[4] S. Narendra, V. De, S. Borkar, D. Antoniadis, and A. Chandrakasan, "Full-chip subthreshold leakage power prediction and reduction techniques for sub-0.18- mu;m cmos," *Solid-State Circuits, IEEE Journal of*, vol. 39, no. 3, pp. 501–510, March 2004.

[5] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor, "Conservation cores: Reducing the energy of mature computations," *SIGARCH Comput. Archit. News*, vol. 38, no. 1, pp. 205–218, Mar. 2010.

[6] N. Goulding-Hotta, J. Sampson, G. Venkatesh, S. Garcia, J. Auricchio, P. Huang, M. Arora, S. Nath, V. Bhatt, J. Babb, S. Swanson, and M. Taylor, "The greendroid mobile application processor: An architecture for silicon's dark future," *Micro, IEEE*, vol. 31, no. 2, pp. 86–95, March 2011.