NVIDIA Charts Its Own Path to ARMv8

While the Tegra line of mobile processors has used the 32-bit ARM Cortex line of CPU IP cores to date, NVIDIA announced back in 2011 that it had taken an architecture license for the 64-bit ARMv8 instruction set and was building a custom ARM core. The result is Project Denver. The company had been quietly building a design team to chart its own unique path in CPU design prior to the announcement. NVIDIA (finally) gave the outside world the first peek at Project Denver's architecture at Hot Chips 26.

The first product to feature the Denver CPU is the Tegra K1-64. This new chip is sampling and like the Tegra K1-32 (already in production), it uses NVIDIA's Kepler GPU and both are socket compatible. The performance of the Denver core is high enough that NVIDIA is comfortable with only two Denver cores instead of four Cortex-A15 cores.

With the larger caches and wide issue operation, Denver can achieve IPC rates comparable to an Intel Core processor, but with the power of a mobile processor. To achieve this mobile chip nirvana, the chip architects at NVIDIA built a unique CPU unlike any other in the ARM ecosystem. It's a big gamble for the company and we're about to see if it paid off.

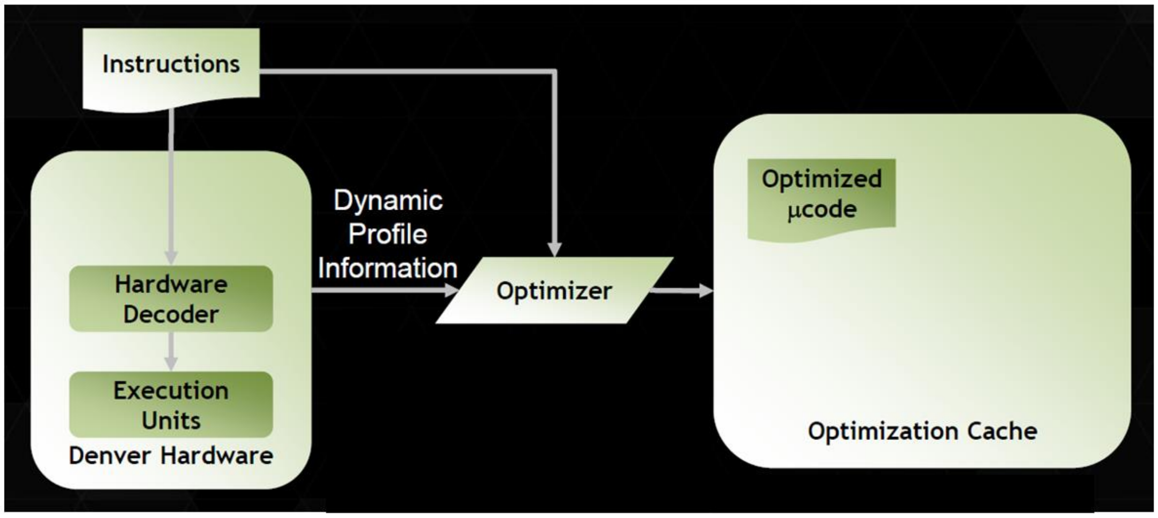
Dynamic Code Optimization

The most unique aspect of Denver is the dynamic code optimization. The core microarchitecture of the CPU is unique in that it has an in-order pipeline, but uses special software to reorder and optimize instruction traces. During repetitive code sequences, the Denver CPU collects dynamic runtime information during code execution and passes this information to the dynamic code optimizer; enabling the optimizer to assess more optimized ways for the code to be executed. The CPU uses hidden time slices to run the optimizer or can use the second core for optimizations for the active core.

The dynamic optimizer runs in its own private and protected state and is not visible to the operating system or any user code. The signed and encrypted dynamic optimizer code loads at boot into a protected part of main memory. By performing the reordering and register renaming in software, Denver eliminates the power hungry out-of-order control logic and yet it can achieve comparable results.

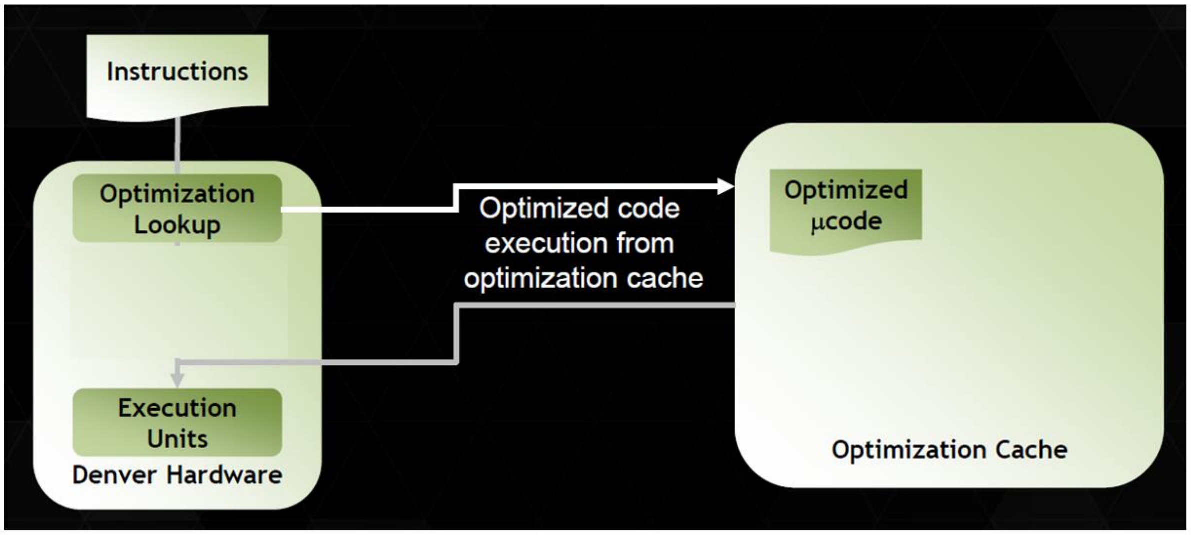
The profiler gathers info on program flow such as branch results (such as taken, not taken, strongly taken, and strongly not taken) and other hardware statistics tables and counters. The optimizer (Figure 1) recognizes opportunities to improve execution and then can rename registers, reorder loads and stores, improve control flow, remove redundant code, hoist redundant computations, perform loop unrolling, and other common optimizations. Because the run-time software performs optimization, the profiler can look over a much larger instruction window than is typically found in hardware out-of-order (OoO) designs. Denver could optimize over a 1,000 instruction window, while most OoO hardware is limited to a 192 instruction window or smaller. The dynamic code optimizer will continue to evaluate profile data and can perform additional optimizations on the fly.

FIGURE 1. DENVER DYNAMIC CODE OPTIMIZER CREATES OPTIMIZED MICROCODE IN CACHE



Once the ARM code sequence is optimized, the new microinstruction sequence is stored in an optimization cache in main memory and can be recalled when the ARM code branch target address is recognized in a special optimization lookup table on-chip. A branch target hit in the table provides a pointer in the optimization cache for the optimized sequence, and that sequence is substituted for the original ARM code (Figure 2). This optimization lookup table is a 1K 4-way memory which holds ARM branch to optimization target pointer pairs, and it is looked up in parallel to the instruction cache. The optimization cache is stored in a private 128MB main memory buffer, a minor set aside in systems with 4GB (or more) of main memory. The optimization cache also does not contain any pre-canned code substitutions design to accelerate benchmarks or other applications.

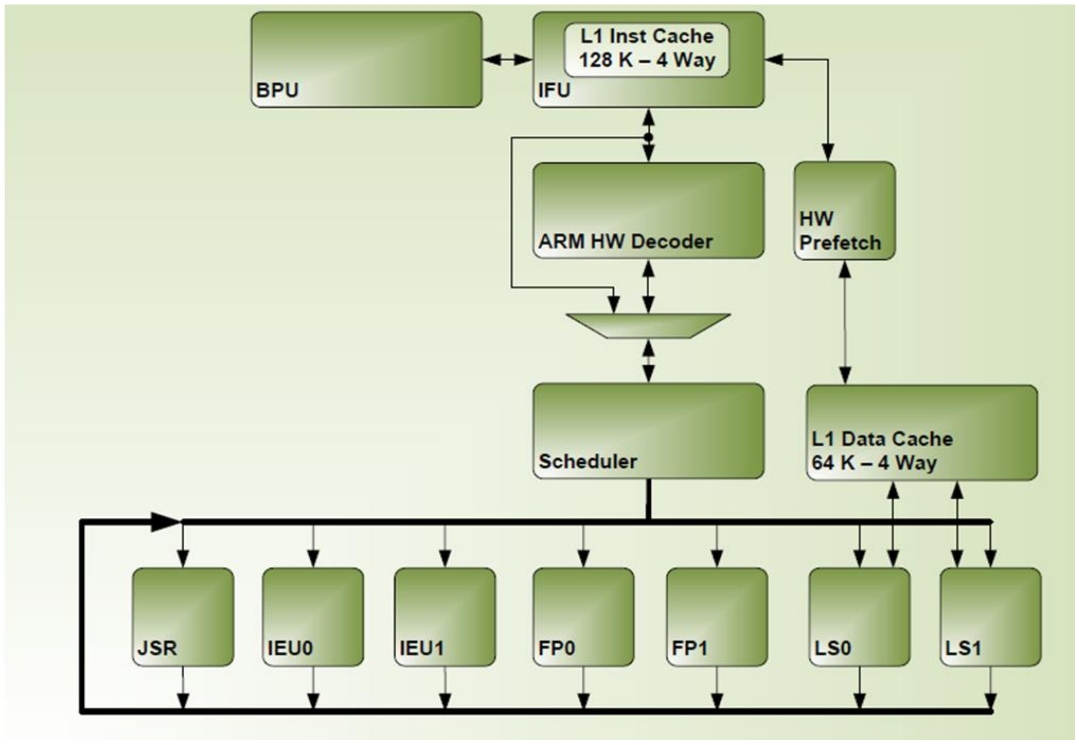
FIGURE 2. SUCCESSFUL OPTIMIZATION LOOKUP GENERATES POINTER TO OPTIMIZED MICROCODE IN CACHE



Based on some code runs shown at Hot Chips, the processor quickly substituted the optimized code after a few runs executed. The optimized code does incur some code expansion, which is why Denver has a larger 128KB L1 instruction cache. The optimized code segments can also be chained in the optimization cache so the optimization lookup cache doesn't need to be accessed for every segment.

Compared with the three instruction decoder of the Cortex-A15 and Cortex-A57, the Denver CPU can execute up to 7+ ARM instructions per cycle using optimized microcode, delivering higher instruction per cycle (IPC) performance on frequently-used code segments. The instruction cache can deliver a 32-byte "parcel" to the scheduler every cycle. The scheduler breaks the parcel into variable sized bundles, and then feeds the functional units with from one to seven operations per cycle. The highest level of parallelism will be generated with optimized code segments. The efficiency of the dynamic code optimization comes from eliminating unnecessary instructions and reordering the code to pack more parallel execution into each bundle.

FIGURE 3. NVIDIA DENVER CORE MAIN FUNCTIONAL UNITS



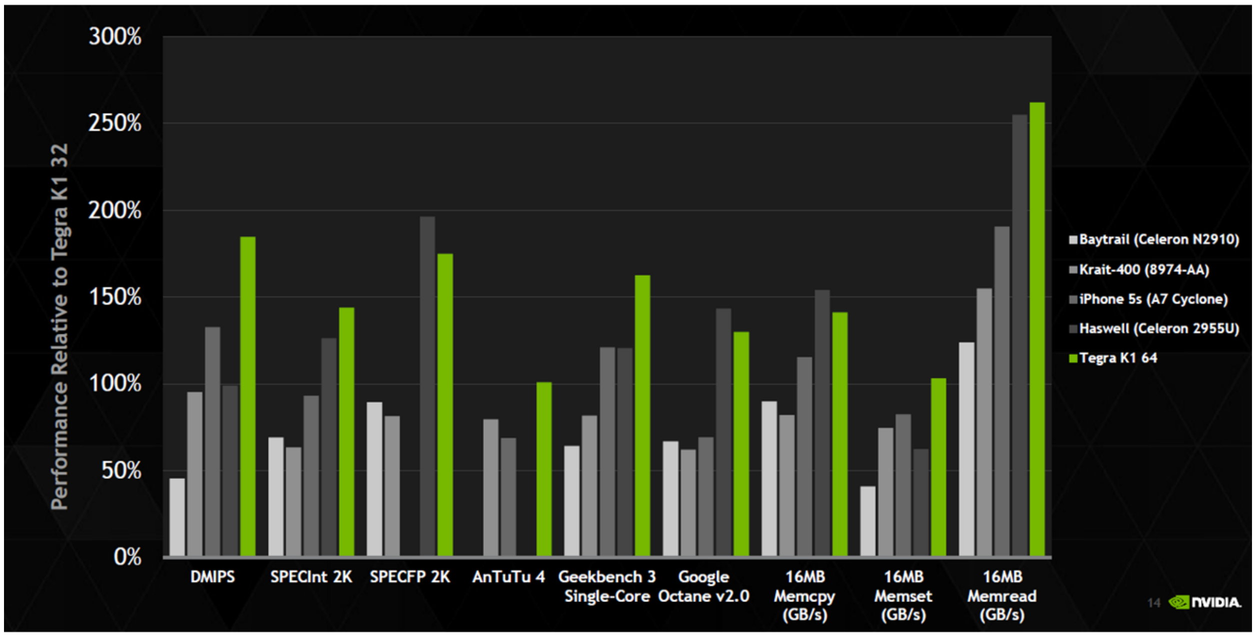
The dynamic code optimizer is run as part of the hardware/firmware privilege level which is not visible to the operating system or hypervisor software. The dynamic code optimization thread can preempt foreground programs or can run on the second core in the Tegra K1-64 if it's available. The dynamic code optimizer can be preempted and can save and restore state. The optimizer is self contained, running necessary services such as table management and garbage collection.

The Denver core has a 13 cycle mispredict penalty, which is less than the 15 cycles for the A15, reducing branch penalties. The shorter pipeline does not seem to impact Denver's clock speed scaling as it is expected to launch at 2.5GHz, faster than NVIDIA's Cortex-A15-based processor (Tegra K1-32).

NVIDIA's performance goals for Denver are quite aggressive - it can actually be compared to desktop CPUs, with some benchmarks demonstrating performance on par with an Intel Haswell-based Celeron processor (Figure 4). It's also extremely fast for a mobile processor as seen when compared to the Apple A7 processor on those same benchmarks. The performance levels shown will make Denver a snappy processor, suitable for super-smartphones, tablets, and

Chromebooks. Additional benchmark results are expected once the Tegra K1-64 reaches production.

FIGURE 4. NVIDIA DENVER BENCHMARKS (SOURCE: NVIDIA)

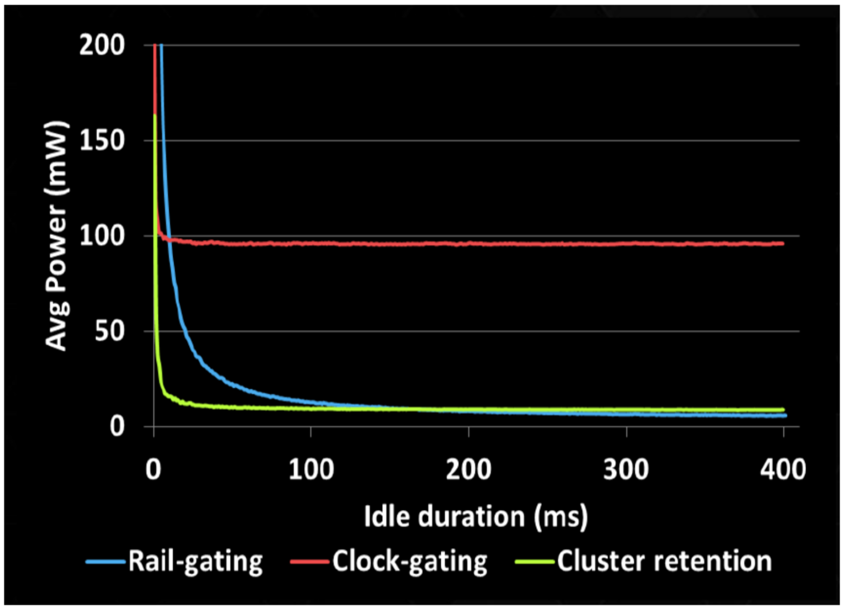


Power Optimized Cores

Because Denver is a custom designed core, NVIDIA could design in its own power savings into the CPU cores. With ARM's fixed CPU cores, such as the Cortex-A15 cores used in the Tegra K1-32, NVIDIA has limited control over the core power management. The company solved the problem by adding a power-optimized fifth core in addition to the four Cortex-A15 cores. This fifth shadow core is activated when only minimal performance is required and the power savings core can run at lower power than the standard cores.

With Denver, NVIDIA had flexibility in the core design when the company took an architecture license for ARMv8 ISA rather than license a Cortex-A57 core. The architecture license allowed it to build a custom microarchitecture, as long as the final processor maintains ARMv8 instruction set compatibility. With the custom design, the company created a new power state that allows the processor to enter a deeper sleep state where logic in turned off, but retains cache and CPU state information. NVIDIA calls this new power state CC4 (Figure 5).

FIGURE 5. DENVER CC4 CLUSTER RETENTION IS FAST AND HAS LOW IDLE CURRENT



The advantage of this new power state is that it improves recovery time by holding the register and state information while still being in a very low power state. The new power state is almost as low as power rail gating the cores, but without the long recovery time. With power rail gating, there's the power and time overhead required to flush the cache contents and saving state before shutting off power. The cache flush overhead and the loss of architecture state adds significant time to the entry and exit of the power state. By maintaining the cache and state information, entering and exiting the CC4 state is faster than power gating, increasing responsiveness while still saving significant power.

Summary:

The 64-bit support combined with extra logic and cache does have a negative side effect - the CPU die area is significantly larger than the Cortex-A15 in the Tegra K1-32. As such, NVIDIA only fit two of the Denver cores instead of the 4+1 A15 cores in the 32-bit part. Denver's peak IPC for optimized code should make up the difference in core count, as NVIDIA showed in some early benchmark data.

The disadvantage of only two cores will be in marketing the Tegra K1-64 against the quad-core and octa-core chips from competitors, but in delivered performance on real-world applications, Denver should do quite well. NVIDIA will have to take a similar approach as Apple in marketing Tegra K1-64 by minimizing references to the number of CPU cores and focus on the strengths of the chip. The company has already taken steps in that direction with the Tegra K1-32, focusing on the 192 CUDA cores in the GPU and ignoring the 4+1 CPU cores.

We expect this is just the first reveal of Denver and NVIIDA has more surprises under the hood that were not revealed at this Hot Chips presentation. The innovations in this chip go deep - it's like peeling an onion and there's still more layers to go.