Haswell metaprogramming

Problem

Some compute-heavy problems or approaches at solving them require too much code in their "inner loops" and/or too many processor registers - for speed

Example: so-called "bitslicing" of cryptographic primitives, where the code resembles a hardware logic circuit, with each data bit requiring a register and each logic gate requiring an instruction

Why do this bitslicing weirdness? Also for speed. Our N-bit CPU becomes N 1-bit CPUs, which provides the sometimes-needed flexibility and lets us optimize away "operations" such as bit shifts, rotates, and permutations.

Another potential example: JIT-compiled code for matching of many requests against many rules in a packet filter, fast HTTP server, anti-DDoS proxy

On typical CPUs such as Intel Haswell and beyond, L1 instruction cache is 32 KiB, the uop cache is smaller yet, and there are 16 general-purpose + 16 SIMD registers per thread. Suppose that's not enough.

Solution

Let the code be in L2 cache, which is 256 KiB

Haswell can fetch and decode 16 bytes of instruction stream per cycle even for code coming from L2 cache (via L1), with no penalty on code throughput. (The 16 bytes limit also applies to fetch & decode from L1 on uop cache miss.)

Haswell can execute up to 3 AVX2 instructions per cycle, and no more of those anyway. If we keep our instructions to <= 5 bytes each, we utilize the AVX2 execution units fully even with code in L2 cache.

Haswell is also fast at accessing L1 data cache (same throughput as registers)

With properly initialized 16 GP registers, we can fit 8-bit immediate offsets against those into 5-byte instructions and thereby address a 16*256 bytes "register file". That's 128 extra AVX2-alike 32-byte "registers".

We got a meta CPU with 256 KiB I-cache and 144 x 256-bit registers. And it's fast. Magic.

The catch

The workload and its portions must be throughput rather than latency bound

Only one of the "registers" in an instruction can be virtual (in L1 cache), and it must be either source-only or destination of a MOV. Other registers in the same instruction have to be among the 16 regular AVX2 registers.

There are only 2 read and 1 write ports to L1 data cache, so on average the 3 instructions processed per cycle must not exceed these ports' capacity. (Yet individual groups of 3 instructions may exceed it with no penalty, as long as the workload is only throughput bound as it should be.)

In testing, this works great for code sizes of up to roughly one half of L2 cache size or slightly more (the threshold is typically at 130 KiB to 140 KiB, although it varies a bit between program invocations). This is likely because of L2 cache being physically-indexed, with higher code sizes resulting in high probability that we'd have duplicate indices in excess of the cache's 8-way associativity. (Page coloring or a custom allocator in the kernel or a kernel module could help run the full 256 KiB of code fast.)