Intel’s Sandy Bridge Microarchitecture

Introduction

Sandy Bridge was once known by the codename Gesher, meaning bridge in Hebrew. While most Intel codenames are geographic in nature, in this case, it can be interpreted as a rather apt metaphor. Sandy Bridge is a synthesis of three separate worlds within Intel – blending the microarchitectures of the Pentium Pro and the Pentium 4 and a new implementation of the GenX graphics architecture. The result is tightly integrated with a novel system infrastructure into a single chip manufactured on Intel’s 32nm process. This article is the first in a series and focuses on the microarchitecture of the Sandy Bridge microprocessor, subsequent articles will deal with the graphics architecture and productizations.

Intel’s first out-of-order microprocessor was the Pentium Pro (P6). It was conceived and designed in Oregon and was also the first Intel CPU solidly targeted at the server market, with glueless SMP support for up to 4 sockets. At the time, servers and workstations were dominated by proprietary RISC architectures, but the Pentium Pro ultimately signaled the decline and eventual death knell of these competitors. Several RISC architectures survive today (notably PPC and SPARC), but they are in a constant race to stay ahead of the x86 ecosystem and no longer enjoy any real growth, but rather are confined to a few lucrative niches.

The P6 first arrived in 1995 on 0.5um and 0.35um BiCMOS processes and was subsequently refined in the consumer oriented Pentium II and Pentium III. At one point, that appeared to be the end of the line for the P6. The successors to the P6 were to be IA64 (for servers) and the Pentium 4 (for client systems). The Pentium 4 was a highly innovative microarchitecture and a radical departure from the previous generation – even more aggressively out-of-order than the P6 and tailored for extreme frequencies. The P4 featured many novel architectural choices such as a trace cache, simultaneous multi-threading and first class SIMD support with SSE 2. Prescott, the second generation of the Pentium 4 microarchitecture eventually hit 3.73GHz in 90nm.

The insatiable desire for high frequencies unfortunately could not overcome the underlying physics – Intel’s single minded pursuit of high frequency lead to high power operation, and there is a firm economic limit of 130W for a mass market product. It is possible to power and cool chips that operate at 250W or even 300W, but the solutions are very expensive and cannot be produced in high volume. Unfortunately, the P4 relied on reaching astronomical frequencies to achieve high performance. AMD took advantage of this weakness (and the delay to embrace a 64-bit x86 extension) and gained substantial market share; for the first time, AMD was a credible and even preferred vendor for server microprocessors. Intel’s management eventually realized that the high frequency and high power approach of the Pentium 4 was a dead end. They turned back to the P6 – which had been refined for low-power notebook applications by the Israeli design team in Haifa over 3 generations (130nm Banias, 90nm Dothan and 65nm Yonah).

The 65nm Merom was the first P6 derivative since 2000 that was intended to span the entire product spectrum, from clients to server. Merom was billed as a merger of the P4 and P6 microarchitectures, but in reality, there was relatively little taken from the P4. Nonetheless, it was a success and Intel began to claw back market share from AMD. The 45nm Nehalem microarchitecture was a further step in the right direction, building on Merom and incorporating simultaneous multi-threading and a new and vastly more efficient system architecture. Nehalem's integrated memory controllers and on-die QuickPath Interconnect were especially critical for the server market. But Westmere, the 32nm shrink of Nehalem, will be the last P6 derivative from Intel. After 15 years, the P6 is finally being replaced by a new microarchitecture: Sandy Bridge.

The Sandy Bridge CPU cores can truly be described as a brand new microarchitecture that is a synthesis of the P6 and some elements of the P4. Although Sandy Bridge most strongly resembles the P6 line, it is an utterly different microarchitecture. Nearly every aspect of the core has been substantially improved over the previous generation Nehalem. Many of these changes, such as the uop cache or physical register files, are drawn from aspects of or concepts behind the P4 microarchitecture. While the P4 was ultimately a flawed implementation, it embodied many good ideas – ideas that are reappearing across the industry, and in Sandy Bridge. The underlying philosophy of Intel's approach to CPU design is to focus on maximizing per-core performance and efficiency. This is a contrast to AMD's Bulldozer, which backs off slightly from per-core performance to emphasize aggregate throughput. This article will explore the microarchitecture of Sandy Bridge – a 64-bit, quad-core, dual threaded, 4 issue, out-of-order microprocessor with the new 3 and 4 operand AVX instruction set extension, implemented in Intel's 32nm process.

System Architecture

As Moore's Law has given more transistors with each generation, the level of achievable integration on microprocessors has steadily climbed. Intel came rather late to the integration game – it wasn't until the 45nm Nehalem in 2008 that the memory controllers and coherent interconnects moved into the CPU silicon. However, as befitting the company that lives by Moore's Law, they have come to the integration game with a vengeance. The 32nm client versions of Westmere packaged the microprocessor with a companion die that included graphics and a memory controller and other features; the two chips were connected via QPI. Sandy Bridge takes this to the logical conclusion and integrates almost everything found in the previous generation into a single 32nm die and eliminates the QPI link. Sandy Bridge can be divided into three major areas – the CPU cores and L3 cache, the GPU and the system agent (previously called the 'uncore'), which holds everything else. The major blocks and external interfaces are shown below in Figure 1. The frequencies for Sandy Bridge in Figure 1 are estimates for the base clocks on a relatively high-end model. Due to the dynamic voltage and frequency scaling (DVFS) system, the CPU cores and GPU will typically

operate at much higher frequencies. As Intel nears the release of products, the actual clockspeed should firm up more.

Sandy Bridge uses a new 1155-pin socket that is not compatible with the previous generation Arrandale and Clarkdale processors. One reason is that clock generation has changed. Rather than having a discrete clock generator on the motherboard, Intel's 6-Series chipset feeds a single base clock through DMI to the processor, where it is multiplied and distributed across the die. This is another example of integration, although the clock generator is a smaller part of the system than the GPU or memory controllers.

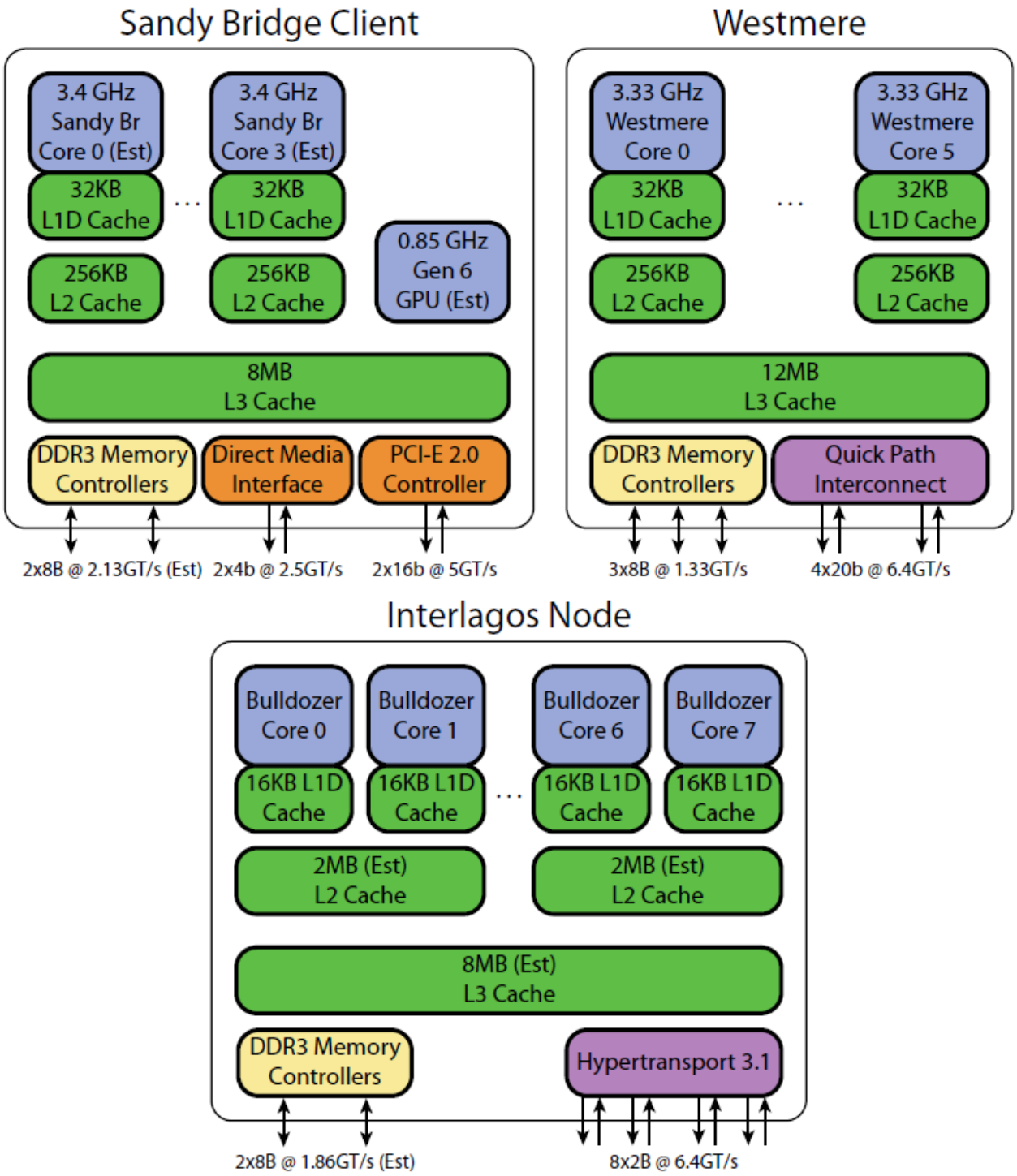


Figure 1 – Sandy Bridge System Architecture and Comparison

Figure 1 above shows the Sandy Bridge client, compared to Westmere and one node in the first Bulldozer implementation (Interlagos). While Interlagos is solidly aimed at servers, at least one variant will appear in the high-end desktop market. It is unclear whether this version will forgo the two node MCM in favor of higher frequencies. When comparing Interlagos or Bulldozer to other

designs, it is important to remember that the front-end, floating point units and L2 cache are shared between two cores in a module.

Sandy Bridge's memory controller should be a bit of an upgrade over the previous generation. Arrandale topped out at 1.066GT/s and the desktop Clarkdale could reach 1.33GT/s. While Intel did not disclose any speeds, it seems reasonable to assume that Sandy Bridge should be able to reach at least 1.86GT/s and quite possibly 2.13GT/s for more extreme systems. On the low power front for notebooks, DDR3L (which operates at 1.35V rather than 1.5V) is now available at 1.33GT/s and will probably be an option. Otherwise, the major interfaces will remain the same.

The Sandy Bridge clients will go into production later this year, with products for 1Q11. This is substantially ahead of AMD's competing Llano processor, which also integrates graphics on-die, but shipments are delayed till 1H11, with products likely a quarter later. Llano is still based on a microarchitecture derived from Barcelona and can be expected to lag significantly in CPU performance. Of course, given AMD's graphics expertise, the GPU in Llano should outperform the Gen 6 graphics in Sandy Bridge. At the low-end, AMD will be shipping derivatives of Bobcat at the same time as Sandy Bridge, but they serve entirely different segments of the market.

Sandy Bridge-EP, the mainstream server version will not arrive till 2H11 – most likely in Q3. Sandy Bridge-EP will retain the same microarchitecture, but with a different system agent. Sandy Bridge-EP is expected to feature 8 cores, 16MB of L3 cache (although some rumors put this at 20MB), 4 DDR3 memory controllers, 2 QuickPath 1.1 links and 32 lanes of PCI-Express 3.0. Naturally, all the extra I/Os require a new socket – the LGA2011, which presumably adds more pins for power and ground. Consumer oriented features like the GPU and display engine will be removed to make room for additional cores, memory controllers and I/O. Sandy Bridge-EP should ship in roughly the same time frame as Interlagos, so server buyers will have the choice between two new microarchitectures. Interlagos will have up to twice as many cores as Sandy Bridge-EP, however, the actual product performance strongly depends on many unknown factors such as frequency and L3 cache design.

Instruction Fetch

The job of the front-end of Sandy Bridge is to consistently deliver enough uops from the instruction stream to keep the back-end occupied. Even the highest performance out-of-order execution CPU will deliver poor results without a capable front-end. For any modern x86 CPU, this is quite challenging. The delivery of the instruction stream is frequently interrupted by branches, and a taken branch may introduce a bubble into the pipeline as instruction fetching is redirected to a new address. Decoding from x86 instruction bytes into uops is complicated by the variable length nature of x86 instructions, the multitude of prefixes, and exceedingly complex microcoded instructions. The Sandy Bridge architects spent tremendous effort to improve all these facets of the front-end. One of the most novel features of the Sandy Bridge microarchitecture is the uop cache, which contains fixed length decoded uops, rather than raw bytes of variable length instructions. A hit in the uop cache

bypasses substantial portions of the front-end and improves the delivery of uops to the back-end. The uop cache is conceptually akin to the trace cache from the Pentium 4, but differs in the details – it has substantially been refined and modified, as we will explore in the next page.

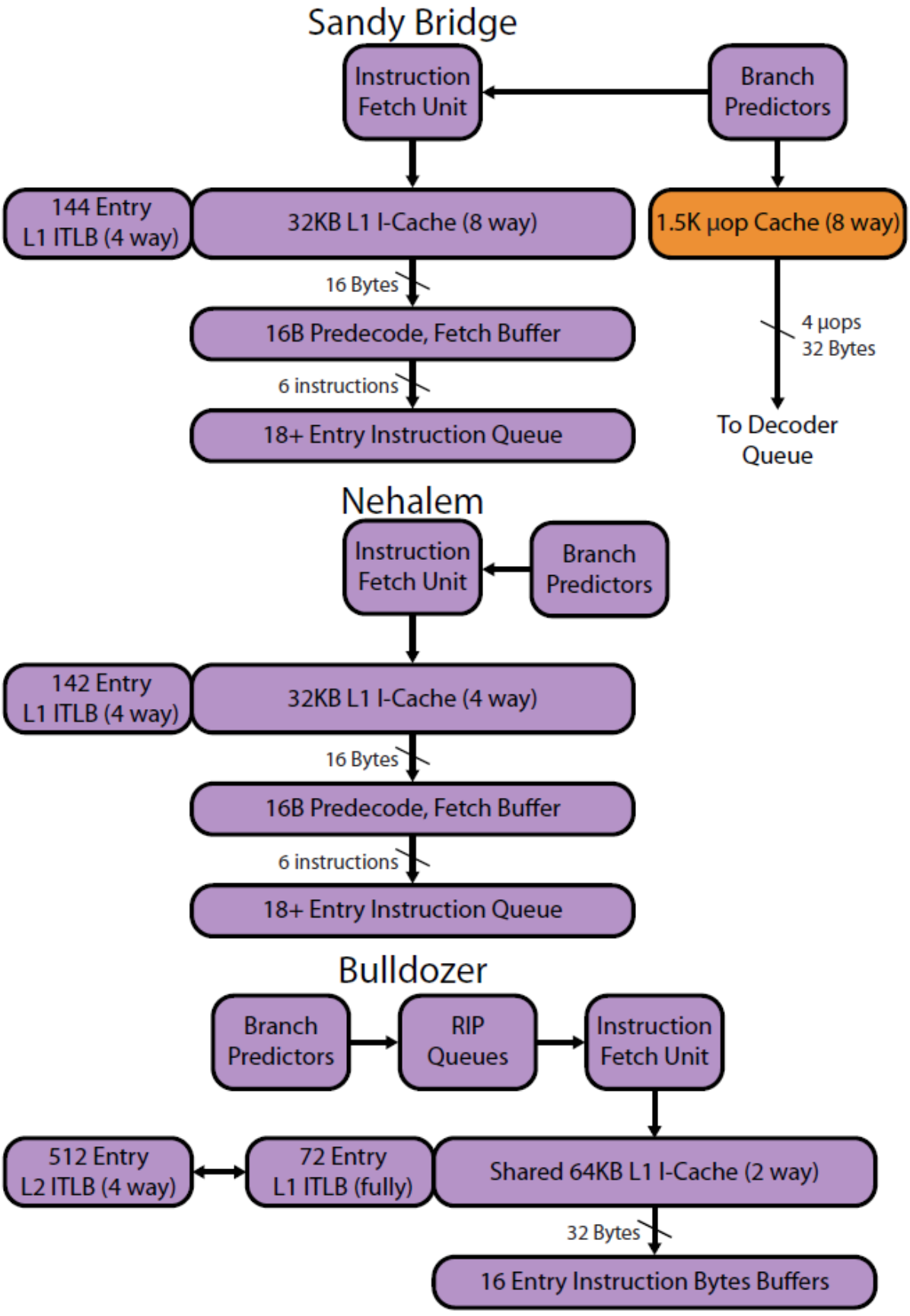

Sandy Bridge
Instruction Fetch Unit
Branch Predictors
144 Entry L1 ITLB (4 way)
32KB L1 I-Cache (8 way)
1.5K µop Cache (8 way)
16 Bytes
16B Predecode, Fetch Buffer
4 µops 32 Bytes
6 instructions
18+ Entry Instruction Queue
To Decoder Queue
Nehalem
Instruction Fetch Unit
Branch Predictors
142 Entry L1 ITLB (4 way)
32KB L1 I-Cache (4 way)
16 Bytes
16B Predecode, Fetch Buffer
6 instructions
18+ Entry Instruction Queue
Bulldozer
Branch Predictors
RIP Queues
Instruction Fetch Unit
512 Entry L2 ITLB (4 way)
72 Entry L1 ITLB (fully)
Shared 64KB L1 I-Cache (2 way)
32 Bytes
16 Entry Instruction Bytes Buffers

Figure 2 – Sandy Bridge Instruction Fetch and Comparison

One of the areas that Intel's microarchitects concentrate on most keenly is branch prediction. It seems like hardly a generation goes by without Intel improving the branch predictors in one fashion or another. The rationale is fairly straight forward. Many improvements that increase performance also increase the energy used; to maintain efficiency, microarchitects must ensure that a new feature gains more performance than it costs in energy or power. In contrast, branch prediction is one of the few areas where improvements generally increase performance and *decrease* energy usage. Each mispredicted branch will flush the entire pipeline, losing the work of up to a hundred or so in-flight instructions and wasting all the energy expended on those instructions. Consequently, avoiding expensive mispredictions with better branch predictors is highly desirable and a prime focus for Intel.

The branch prediction in Sandy Bridge was totally rebuilt for better performance and efficiency, while using the same amount of resources. Sandy Bridge retains the four branch predictors found in Nehalem: the branch target buffer (BTB), indirect branch target array, loop detector and renamed return stack buffer. Sandy Bridge has a single BTB that holds twice as many branch targets as the L1 and L2 BTBs in Nehalem, yielding better branch prediction coverage. The single level design was accomplished by representing branches more efficiently and essentially compressing the number of bits required per branch. For example, any taken branch in the predictor must include the displacement from the current IP; branches with a large displacement can be held in a separate table so that most branches (which have a short displacement) do not require as many bits. While Intel did not disclose the number of targets for Nehalem, the P4 BTB had 4K targets, and it seems reasonable that Sandy Bridge has 8K-16K. Just as importantly, the global branch history, which tracks the most recently predicted (and also executed) branches, increased in size to capture a longer pattern history. Again, the number of bits used did not increase – instead, Intel omits certain branches from the pattern history that do not help to make predictions.

Nehalem enhanced the recovery from branch mispredictions, which has been carried over into Sandy Bridge. Once a branch misprediction is discovered, the core is able to restart decoding as soon as the correct path is known, at the same time that the out-of-order machine is clearing out uops from the wrongly speculated path. Previously, the decoding would not resume until the pipeline was fully flushed.

The instruction fetch for Sandy Bridge is shown above in Figure 2. Branch predictions are queued slightly ahead of instruction fetch so that the stalls for a taken branch are usually hidden, a feature earlier used in Merom and Nehalem. Predictions occur for 32B of instructions, while instructions are fetched 16B at a time from the L1 instruction cache.

Once the next address is known, Sandy Bridge will probe both the uop cache (which we will discuss in the next page) and the L1 instruction cache. The L1 instruction cache is 32KB with 64B lines, and the associativity has increased to 8-way, meaning that it is virtually indexed and physically tagged. The L1 ITLB is partitioned between threads for small pages, with dedicated large pages per thread.

Sandy Bridge added 2 entries for large pages, bringing the total to 128 entries for 4KB pages (for both threads) and 16 fully associative entries for large pages (for each thread).
The instruction fetcher will retrieve 16B from the instruction cache into the pre-decode buffer. The pre-decoder will find and mark the instruction boundaries, decode any prefixes and check for certain properties (e.g. branches). The pre-decoder throughput is limited to 6 instructions per cycle, until the 16B instruction fetch is consumed and the next one can begin. Since the pre-decoding is done in 16B chunks, average throughput can suffer at the end of a chunk. For instance, the first cycle could pre-decode 15B into 4 instructions, leaving 1B and 1 instruction for the second cycle and resulting in overall throughput of 2.5 instructions per cycle. Large immediates can have a similar impact on throughput as well. Once pre-decoded, the instructions are placed into the instruction queue for decode. The size of the instruction queue in Merom was 18 entries, it has almost certainly increased for Nehalem and Sandy Bridge, but the precise value is unknown.

Instruction Decode and uop Cache

Decoding fetched instructions in Sandy Bridge has not changed much, as shown below in Figure 3. The instruction queue can send 4 pre-decoded instructions (or 5 with macro-fusion on a compare and branch) to the x86 decoders. The decoders read in the x86 instructions and emit regular, fixed length uops which are natively processed by the underlying hardware. The four decoders are still asymmetric; the first decoder can handle complex instructions that emit upto 4 uops and the other three decoders all handle simpler instructions. For microcoded instructions (i.e. >4 uops), the microcode will decode 4 uops/cycle until the instruction is finished – although this blocks regular decoding. In both Sandy Bridge and Nehalem, the decoders can emit at most 4 uops per cycle – no matter what mix of instructions are being decoded.
Most of the new 256-bit AVX instructions in Sandy Bridge are treated as simple instructions by the decoders, thanks to some rather clever implementation work in the execution units and memory pipeline. As with previous generations, the decoders all support uop micro-fusion so that memory-register instructions can decode as one uop. The dedicated stack pointer tracker is also present in Sandy Bridge and renames the stack pointer, eliminating serial dependencies and removing a number of uops. Once instructions are decoded, the uops are sent to the uop cache and to the back-end for allocation, renaming and out-of-order execution. Subsequent instruction fetches to the same address should hit in the uop cache, rather than relying on the normal instruction fetch and decode path.

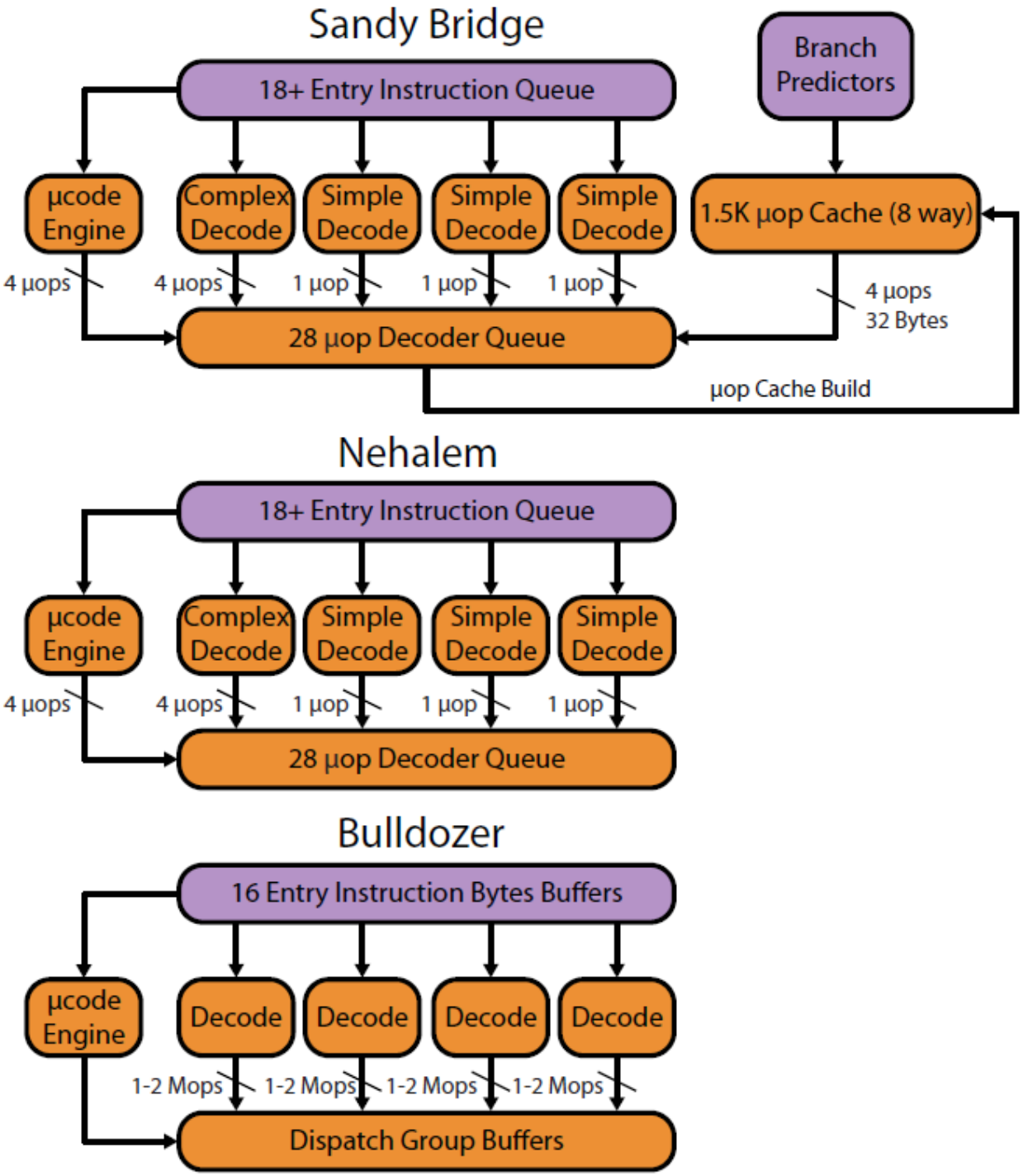


Figure 3 – Sandy Bridge Instruction Decode and Comparison

The most interesting part of Sandy Bridge's front-end is the new uop cache, which promises to improve performance and power consumption. Sandy Bridge's uop cache is conceptually a subset of the instruction cache that is cached in decoded form, rather than a trace cache or basic block cache. The uop cache shares many of the goals of the P4's trace cache. The key aims are to improve the bandwidth from the front-end of the CPU and remove decoding from the critical path. However, a uop cache is much simpler than a trace cache and does not require a dedicated trace BTB or

complicated trace building logic. In typical Intel fashion, the idea first came back as an instruction loop buffer in Merom, then a uop loop buffer in Nehalem and finally a full blown uop cache in Sandy Bridge – a consistent trend of refinement.

In Sandy Bridge, the instruction stream is divided into a series of windows. Each window is 32B of instructions, possibly limited by any taken branches within the window. The mapping between the instruction cache and uop cache occurs at the granularity of a full 32B window. After an entire window has been decoded and sent to the back-end, it is also inserted into the uop cache. Critically important, this build process occurs in parallel with the normal operation of the front-end and does not impose any delays.

Lines in the uop cache are addressed by the IP of the first decoded x86 instruction stored in the line, so the cache is virtually addressed. The entries are tagged so that two threads can statically partition the uop cache. As an added benefit, context or VM switches do not cause a flush (as they did for the trace cache). Since the uop cache works at the granularity of decoded 32B windows, the pseudo-LRU eviction policy must evict all the uop lines corresponding to a given 32B window at once. Self modifying code may also cause evictions, but does not invalidate the whole uop cache.

Sandy Bridge's uop cache is organized into 32 sets and 8 ways, with 6 uops per line, for a total of 1.5K uops capacity. The uop cache is strictly included in the L1 instruction cache. Each line also holds metadata including the number of valid uops in the line and the length of the x86 instructions corresponding to the uop cache line. Each 32B window that is mapped into the uop cache can span 3 of the 8 ways in a set, for a maximum of 18 uops – roughly 1.8B/uop. If a 32B window has more than 18 uops, it cannot fit in the uop cache and must use the traditional front-end. Microcoded instructions are not held in the uop cache, and are instead represented by a pointer to the microcode ROM and optionally the first few uops.

According to Intel, the uop cache performs like a 6KB instruction cache and has a roughly 80% hit rate. By comparison, the 12K uop trace cache of the P4 was supposed to have similar performance to a 8KB-16KB instruction cache. The 4-8X difference in effective storage between the two designs is driven by Sandy Bridge's more powerful uops and eliminating duplicate entries, which plagued the P4's trace cache.

Going back a bit to the instruction fetch, the predicted IP is used to probe the tags for the uop cache in parallel with the regular instruction cache. A hit in the uop cache tags yields set and way identifiers that are put into a match queue and then used to access the uop cache data arrays. When a hit occurs, the uop cache will retrieve all of the uops in a 32B window (i.e. up to 3 lines) into a nearby queue or buffer. It is quite likely that the uop cache data arrays only read out a single line each cycle, so a hit may take multiple cycles to finish. Like the decoders, the uop cache can send up to 4 uops per cycle to the Decoder Queue. However, a uop cache hit can span a full 32B window of instructions. This doubles the bandwidth of the traditional front-end, which is limited to 16B instruction fetches. The extra bandwidth is particularly helpful where the average instruction length is over 4 bytes; for example, AVX instructions use a 2 or 3 byte prefix. Additionally, since each uop cache line can hold more uops than the back-end can rename and allocate (6 vs. 4), the queuing

within the uop cache can hide the pipeline bubbles introduced by taken branches and effectively ‘stitch across taken branches’.

A hit in the uop cache will completely bypass and clock-gate the instruction fetch and decode hardware – saving power and improving the front-end uop supply. After a hit, the uop cache calculates the next sequential IP using the instruction length fields that are stored in each uop cache line. As mentioned above, taken branches are handled by the traditional branch predictor. In the case of a uop cache miss, then the instruction fetch and decode proceed as previously described and the uop cache can be clock gated to save power.

The Sandy Bridge architects decided that handling partial hits in the uop cache is too complex and power inefficient. Resolving partial hits would require activating both the traditional front-end and the uop cache, plus additional logic for synchronizing the two partial outputs. The uop cache line placement and eviction policies were designed so that each 32B window is fully cached (or not at all). Thus when a hit occurs, the traditional front-end is not involved.

All the uops from the traditional front-end and the uop cache are ultimately delivered into the 28 uop Decoder Queue. The Decoder Queue still acts as a cache for small loops like in Nehalem, and actually can bypass both the instruction cache and uop cache.

One of the critical differences between the uop cache in Sandy Bridge and the P4’s trace cache is that the uop cache is fundamentally meant to augment a traditional front-end. In contrast, the Pentium 4 attempted to use the trace cache to replace the front-end and relied on vastly slower fetch and decode mechanisms for any workloads which did not fit in the trace cache. The uop cache was carefully designed so that it is an enhancement for Sandy Bridge and does not penalize any workloads.

The uop cache is one of the most promising features in Sandy Bridge because it both decreases power and improves performance. It avoids power hungry x86 decoding, which spans several pipline stages and requires fairly expensive hardware to handle the irregular instruction set. For a hit in the uop cache, Sandy Bridge’s pipeline (as measured by the mispredict penalty) is several cycles shorter than Nehalem’s, although in the case of a uop cache miss, the pipeline is about 2 stages longer. The uop cache increases performance by more consistently delivering uops to the back-end and eliminating various bubbles in the fetch and decode process. For example, the 16B fetch, length changing prefixes and the decoding restrictions all limit the traditional front-end, whereas the uop cache ignores those issues and can achieve higher performance. Generally, the uop cache seems to avoid the problems of a trace cache, while delivering most of the benefits in a much more power efficient manner.

Out-of-Order Execution

Sandy Bridge’s out-of-order execution is one of the areas where the marriage of the P6 and P4 microarchitectures can be seen most clearly. Like the P6, Sandy Bridge generally treats integer,

floating point and SIMD uops in a unified manner. This is a direct contrast to AMD's various microarchitectures, where floating point and SIMD are handled by a co-processor (Bulldozer) or a separate cluster. Sandy Bridge uses fundamentally different, and more efficient, techniques for tracking and renaming the uops in flight, as compared to the approach used in P6 derivatives such as Nehalem. Instead, Intel's architects borrowed from and adapted concepts earlier used in the P4 microarchitecture and simultaneously expanded the out-of-order resources substantially.

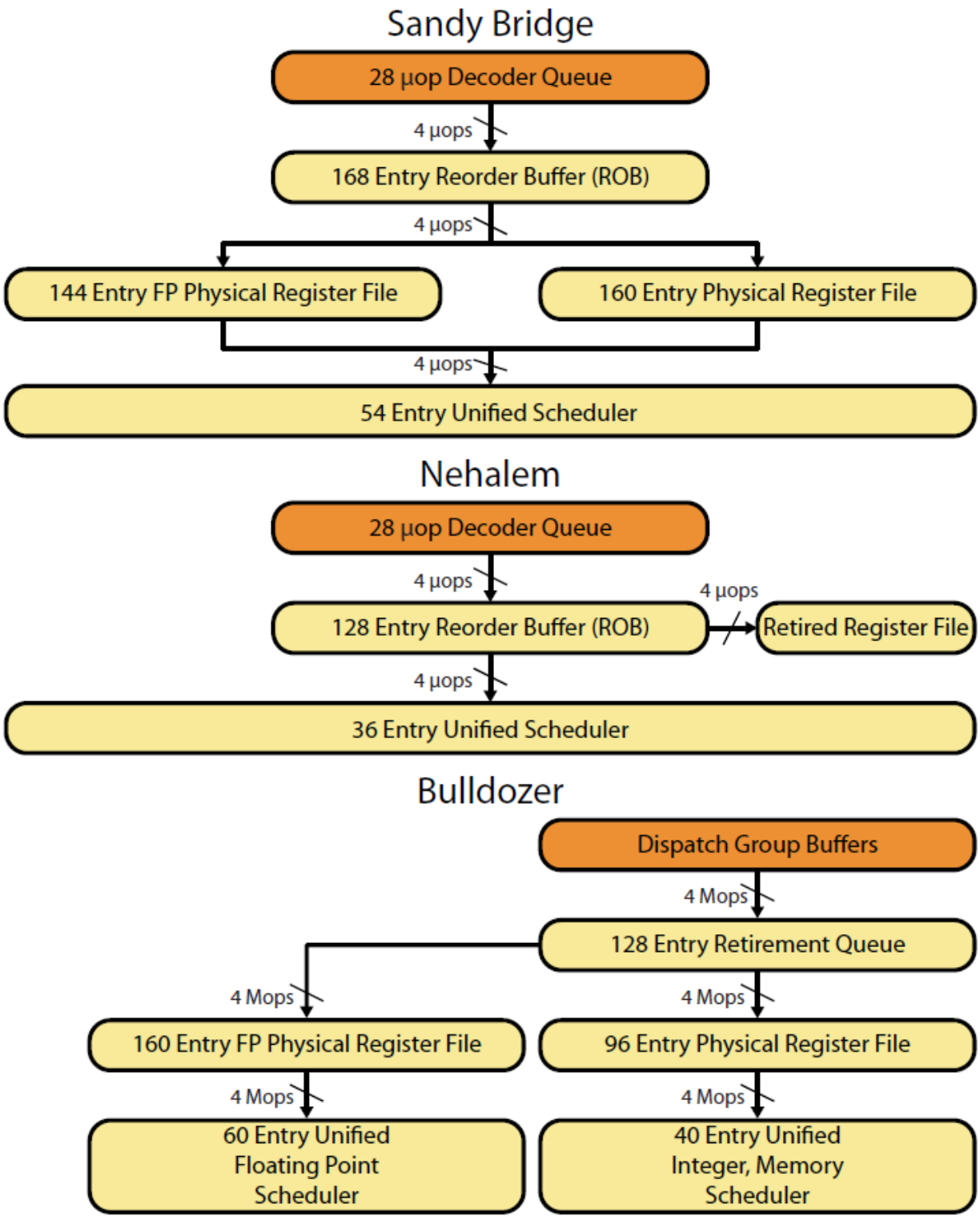


Figure 4 – Sandy Bridge Out-of-Order Execution and Comparison

As shown in Figure 4, Sandy Bridge uses a physical register file (PRF) based renamer. Intel first used this approach for the P4, and AMD has also adopted PRFs for Bulldozer and Bobcat.

Interestingly, almost all high performance out-of-order designs are using the same approach, as IBM uses PRFs for their POWERx line as well.

As described in our Bulldozer article, the earlier P6 design (up to Nehalem and Westmere) used a Re-Order Buffer (ROB) that contained both data and status bits for uops in flight and a separate Retirement Register File (RRF). Retired uops would write their results from the ROB into the RRF. In a PRF design, all data is held in the PRF, a separate structure holds status information and speculative and logical registers are mapped into the PRF. Retirement is handled by simply changing the mapping so that an architectural register points to the correct value in the PRF, rather than moving data.

For Sandy Bridge (and the P4), Intel still uses the term ROB. But it is critical to understand that, in this context, it only refers the status array for in-flight uops (in Bulldozer, this status array is called the Retirement Queue). The two PRFs contain all the data values (both renamed and architectural). A third structure, the Register Alias Tables, maintain the mapping of logical registers (i.e. most recent speculative and also architectural state) to the underlying physical registers in the PRF.

Moving to a PRF based microarchitecture creates a level of indirection and changes many of the critical steps in the out-of-order execution. Allocation, renaming, scheduling and retiring are all different for Sandy Bridge, and minimize the movement and replication of data within the processor. Of course, indirection is not free and probably adds an extra pipeline stage of latency, but the benefits are worth the trade-off.

PRF-based renaming and scheduling is substantially more power efficient because it eliminates movement of 32, 64, 128 or 256-bit data values and instead relies on pointers and changing the vastly smaller mapping tables. Intel's architects used these power savings to substantially enlarge the out-of-order resources in the core and increase per-core performance. Sandy Bridge's ROB was increased to sustain 168 uops in-flight, up from 128 in Nehalem. The number of rename registers more than doubled, partially because some uops may require multiple rename registers (e.g. a register, memory uop may needs an integer rename register for the load address and an separate register for the result). Sandy Bridge's integer PRF holds 160 distinct 64-bit values, while the FP PRF is 144 entries and 256-bits wide to accommodate the full YMM registers in AVX. The scheduler expanded to hold 54 uops and the memory ordering buffers also substantially increased in size.

When using AVX, the architectural state of an x86 thread increases substantially. The YMM registers hold 256B more data than the XMM registers, which creates extra work when executing a SAVE or RESTORE instruction for a context or VM switch. In Sandy Bridge, the hardware tracks which registers are used during execution and will selectively save and restore only the registers that are actually used. Given that the basic register state for x86-64 is slightly over 700B (including 64-bit GPRs, x87 and YMM), this can be quite helpful. For example, a thread only using the 16 GPRs would eliminate moving roughly 600B of data on a context switch.

Another small benefit of a PRF-style design is zeroing registers. A common idiom in x86 code is clearing a register by XORing it with itself, to break any dependencies. In previous generations, this required a uop in one of the ALUs to overwrite the register. However, with a PRF-based design,

zeroing a register can be accomplished solely within the renamer – by simply adding the register to the list of freely available registers in the PRF. This trick may also be used with VZEROALL instructions in AVX.

Sandy Bridge's front-end can deliver 4 uops per cycle from one of the two threads. The uops are still processed in-order at this point, and allocate the necessary resources to track execution. Each uop will allocate a sequential entry in the ROB, to track status and completion information and maintain the correct program order. Integer uops require an available PRF entry to rename their output, while FP and SIMD uops use the FP PRF. Any uops that access memory must also allocate an entry in the appropriate load or store buffer. As previously mentioned, 256-bit AVX instructions decode into a single uop and will not consume extra ROB or PRF entries.

Once uops have been allocated and renamed, they can freely execute out-of-order. Sandy Bridge uses a unified scheduler that is dynamically shared between threads, like Nehalem, but nearly twice the capacity. This has the advantage of allowing a more flexible mix of instructions to execute efficiently compared to distributed schedulers. Renamed uops are entered into the 54 entry unified scheduler, where they wait until they are ready to execute. When a uop is ready, it will be issued to the appropriate execution units. Like Nehalem, Sandy Bridge can issue up to 6 uops to different ports and retire 4 uops per cycle.

Execution Units

The execution units in Sandy Bridge were reworked to double the FP performance for vectorizable workloads by efficiently executing 256-bit AVX instructions. Almost all 256-bit AVX instructions are decoded into and execute as a single uop – in contrast to AMD's more cautious embrace of AVX, which will crack 256-bit instructions into two 128-bit operations on Bulldozer. When Intel introduced SSE2 in the P4, each 128-bit instruction was cracked into two 64-bit uops, and the throughput did not substantially improve. This created a chicken and egg problem: Intel wanted developers to use SSE2 (since the P4 was not designed to execute x87 particularly fast), but developers do not want to rewrite or recompile code for a marginal gain. This is one reason why it took 5-8 years for SSE2 to truly become pervasive.

Sandy Bridge can sustain a full 16 single precision FLOP/cycle or 8 double precision FLOP/cycle – double the capabilities of Nehalem. This guarantees that software which uses AVX will actually see a substantial performance advantage on Sandy Bridge and should spur faster adoption. Intel seems to have learned from the lessons of SSE2 and hopefully, the uptake for AVX amongst the software community will be far swifter.

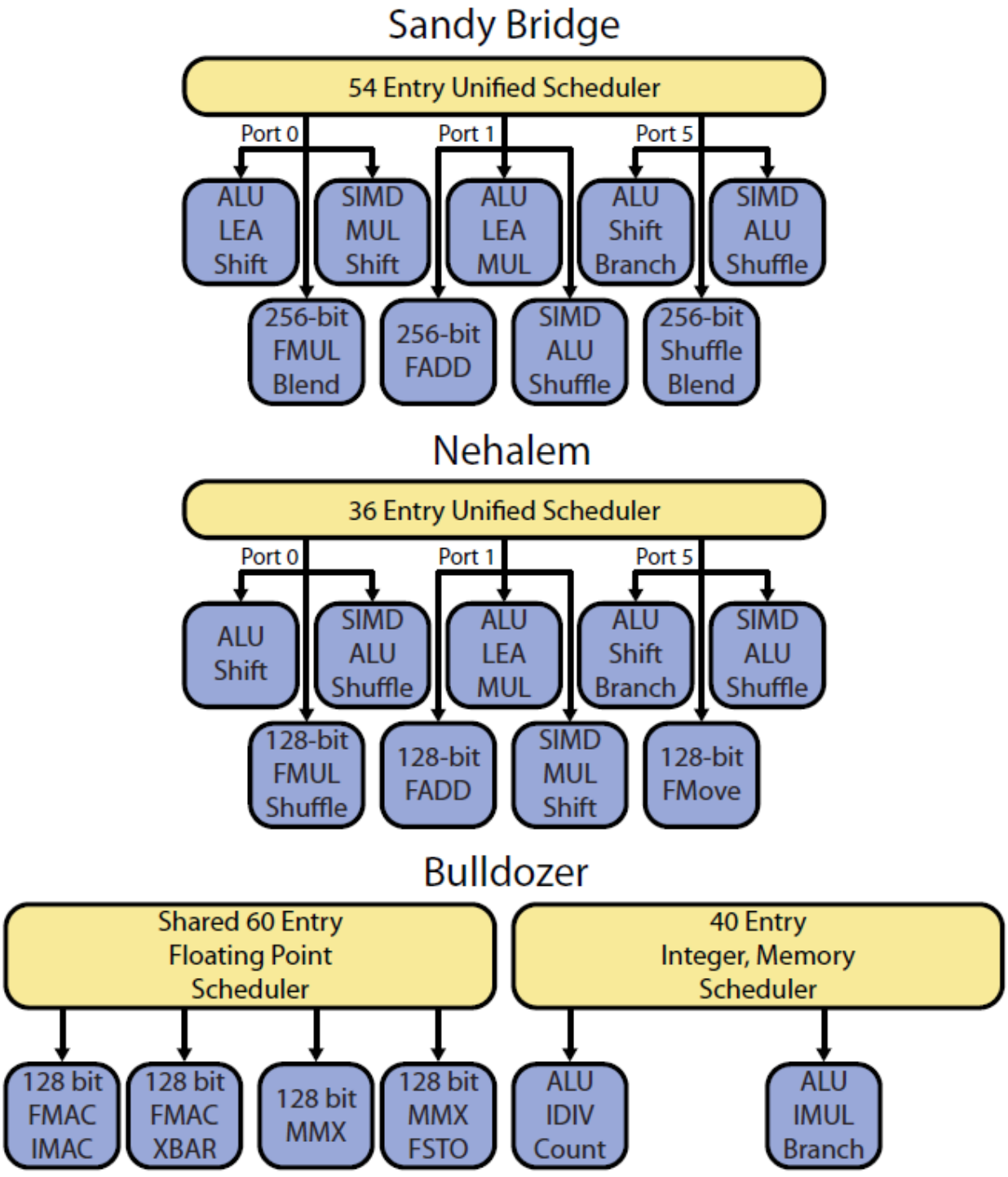


Figure 5 – Sandy Bridge Execution Units and Comparison

As Figure 5 above indicates, Sandy Bridge can execute a 256-bit FP multiply, a 256-bit FP add and a 256-bit shuffle every cycle. However, the floating point data paths were not expanded and are still 128-bits wide; instead the SIMD integer data paths are enlisted to assist with AVX operations. Understanding this technique requires a slight detour and further exploration of the execution units.

The scheduler will issue the oldest, ready to execute uops each cycle to the six available ports, depending on the type of uop. Ports 0, 1 and 5 are used for executing computational uops. There are three types of computational uops and execution units or execution stacks: integer, SIMD integer (which we will refer to as SIMD) and FP (either scalar or SIMD). Each of the 3 execution ports has hardware for the 3 different types of uops. For example, on Sandy Bridge the FP stack for port 1 can execute a 128-bit FP add, and the SIMD stack can execute a 128-bit SIMD add. Each of the three execution stacks is considered to be a different 'domain'. There is free bypassing within each domain, but a 1-2 cycle penalty for bypassing between domains (e.g. an integer uop that receives a forwarded result from an FP uop). This technique is primarily helpful to save power and reduce the complexity of the forwarding network for rarely used cases. The memory pipeline and ports 2-4 are within the integer domain, since integer uops are the most latency sensitive.

Instead of widening the data paths to 256-bits, the Sandy Bridge architects moved the integer SIMD stacks to slightly different issue ports and cleverly re-use the existing 128-bit SIMD and 128-bit FP data paths ganged together to execute 256-bit uops. For example, a 256-bit multiply can issue to port 0 and simultaneously use the 128-bit SIMD data path for the low half and the 128-bit FP data path for the high half. This technique requires some extra logic, but it saves substantial area and power, by re-using execution resources that are already present. The 256-bit shuffle on port 5 also requires dedicated hardware for crossing between the two 128-bit lanes. Fortunately, all the extra logic to re-use the SIMD execution units is relative small and power efficient compared to the area and leakage necessary to double the data paths.

Figure 5 above shows the major uop types for each execution unit, but due to space constraints some are omitted from the diagram. At a high level, Sandy Bridge makes some improvements to the integer execution units, moves around some of the SIMD units and enhances the FP units. On the integer side, there is a new second port for LEA. For SIMD integer, port 0 and port 1 have effectively swapped places (compared to Nehalem/Westmere) and port 5 gained a shuffle. Note that in both designs, PSAD and string uops map to the same port as the integer SIMD multiply. There are also integer blends on ports 0 and 5. On the floating point side, the execution width doubled, the shuffle moved to port 5 and blends were added to ports 0 and 5. Both Nehalem and Sandy Bridge have FP moves in ports 0 and 5 as well. For those interested in the full details, the Sandy Bridge optimization manual should provide a comprehensive description when it arrives.

Sandy Bridge also improves performance for certain security primitives, such as the microcoded AES instructions that were added with Westmere and large number arithmetic. Sandy Bridge improves SHLD (shift left double) performance, which is used for SHA-1 hashing. The throughput for ADC (add with carry) doubled, which is used for large number routines calculations such as RSA.

Memory Subsystem

Since Sandy Bridge can potentially double the FP performance compared to Nehalem, the memory pipeline must substantially improve to keep pace and feed the execution units. This necessitated a redesign of the memory pipeline, particularly the elements closest to the core. When the Sandy Bridge architects were evaluating how to improve the memory pipeline, one of the key goals was to ensure that the changes would benefit all software, not just AVX. Even though the performance for AVX should be good enough to encourage adoption, it will still take time to penetrate the software world. Given these constraints, and the focus on power efficiency, the design point is fairly straight forward.

To start with, the number of memory operations in-flight increased dramatically, in tandem with the overall instruction window. The load buffer grew by 33% and can track 64 uops in-flight. Sandy Bridge's store buffer increased slightly to 36 stores, for an overall 100 simultaneous memory operations, roughly two thirds of the number of the total uops in-flight. To put this in perspective, the number of memory uops in-flight for Sandy Bridge is greater than the entire instruction window for the Core 2 Duo. Again, like Nehalem, the load and store buffers are partitioned between threads.

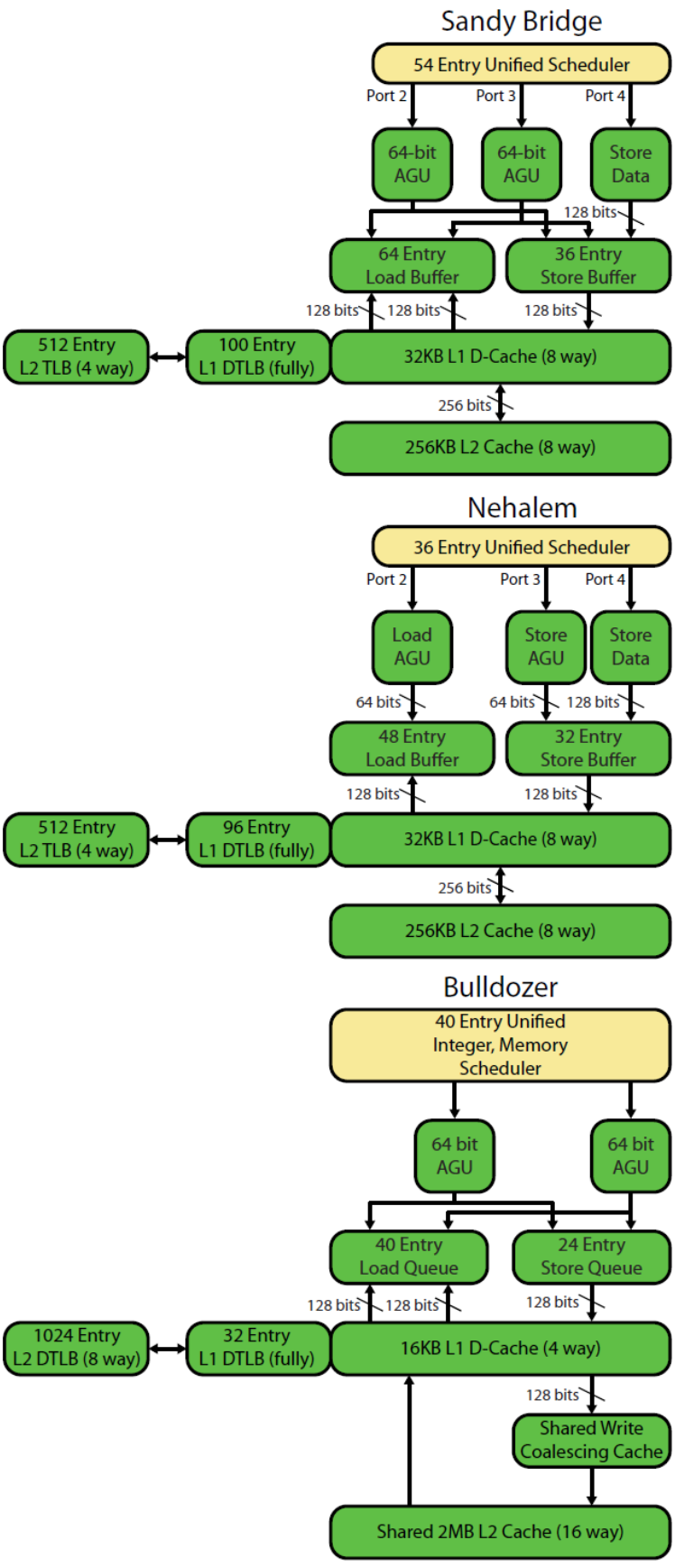

Sandy Bridge
54 Entry Unified Scheduler
Port 2
Port 3
Port 4
64-bit AGU
64-bit AGU
Store Data
128 bits
64 Entry Load Buffer
36 Entry Store Buffer
128 bits
128 bits
128 bits
512 Entry L2 TLB (4 way)
100 Entry L1 DTLB (fully)
32KB L1 D-Cache (8 way)
256 bits
256KB L2 Cache (8 way)
Nehalem
36 Entry Unified Scheduler
Port 2
Port 3
Port 4
Load AGU
Store AGU
Store Data
64 bits
64 bits
128 bits
48 Entry Load Buffer
32 Entry Store Buffer
128 bits
128 bits
512 Entry L2 TLB (4 way)
96 Entry L1 DTLB (fully)
32KB L1 D-Cache (8 way)
256 bits
256KB L2 Cache (8 way)
Bulldozer
40 Entry Unified Integer, Memory Scheduler
64 bit AGU
64 bit AGU
40 Entry Load Queue
24 Entry Store Queue
128 bits
128 bits
128 bits
1024 Entry L2 DTLB (8 way)
32 Entry L1 DTLB (fully)
16KB L1 D-Cache (4 way)
128 bits
Shared Write Coalescing Cache
Shared 2MB L2 Cache (16 way)

Figure 6 – Sandy Bridge Memory Subsystem and Comparison

In Nehalem, there are two address generation units. One is dedicated for load uops (on port 2), while the other can only be used for store uops (on port 3). As another example of re-using existing resources in a more efficient fashion, Sandy Bridge retains the two address generation units. However, they are fully flexible now and each AGU can be used for loads or stores, effectively doubling the load bandwidth. This is a huge benefit, since many workloads can take advantage of two loads per cycle, whereas a 1:1 ratio of loads to stores is relatively unusual.

After address generation, uops will access the DTLB to translate from a virtual to a physical address, in parallel with the start of the cache access. The DTLB was mostly kept the same, but the support for 1GB pages has improved. Previously, Westmere added support for 1GB pages, but fragmented 1GB pages into many 2MB pages since the TLB did not have any 1GB page entries. Sandy Bridge adds 4 dedicated entries for 1GB pages in the DTLB.

Sandy Bridge's L1D cache was redesigned to maintain the same low latency for existing workloads, while increasing bandwidth to match the compute performance gains when using AVX. Sandy Bridge's 32KB L1D is 8-way associative, meaning that it is virtually indexed and physically tagged, and uses 64B lines and a write-back policy. The load-to-use latency is steady at 4 cycles for integer uops (due to bypass, the latency for FP and SIMD is 1 or 2 cycles more). The L1D cache can sustain two 128-bit loads and a 128-bit store every cycle, a 50% increase in bandwidth. Accordingly, the L1D cache has multiple 8 byte banks for simultaneous accesses. When a bank conflict occurs, one of the two memory operations will be delayed a cycle and a state machine will record the conflict and adjust scheduling in the future to prevent subsequent bank conflicts. Both load ports are equipped to efficiently handle misaligned memory accesses, store forwarding and memory disambiguation. As with Nehalem, there are 10 outstanding misses from the L1D cache to higher levels of the memory hierarchy.

Like Bulldozer, Sandy Bridge cannot sustain the full bandwidth of the cache with 128-bit accesses because there are three data accesses per cycle to the L1D, but only two issue ports for addresses and AGUs. Servicing three data requests per cycle can help to clear out some queued operations, but the main benefit is when executing 256-bit AVX code, where the extra bandwidth is essential to sustaining a 2:1 ratio of loads and stores.

According to Intel, 256-bit memory accesses execute as a single uop, thus they only use one entry in the ROB, PRF and memory order buffers. However the actual data accesses are 128-bits wide, so a clever approach is needed to have a 256-bit load execute as a single uop. The most likely technique is described in an Intel patent application and suggests using the 1 cycle bypass delay to guarantee that both halves of a 256-bit load complete in the same cycle. The high half would be initiated in the first cycle, doing address calculation, a TLB lookup and checking the cache tags for a hit, and the 128-bit high half would reach the FP execution stack 5 cycles later. The low half would subtract 16B from the address of the high half and start in the second cycle, with the result reaching the SIMD execution stack 4 cycles later. Thus both 128-bit accesses would arrive in the same cycle and complete together as a single 256-bit uop. A variation of this might also work for stores, but the

general approach only works for aligned accesses. Crossing a cache line or page would complicate the situation by requiring a separate tag check for a hit or a separate TLB look up.
AVX does not have scatter/gather like a GPU or Larrabee, because the area, power and performance impact would be quite substantial and compete with the key goal of low power and low latency for the L1D cache. However, AVX has conditional masking for each lane of vector loads and stores. Suppressed lanes in a memory operation will not raise any faults (e.g. page faults or protection violations). This is much less overhead than scatter/gather, since much of the logic is already present for handling misaligned data or store forwarding. AVX also includes broadcast instructions to all lanes of a vector.
The unified, per-core L2 cache in Sandy Bridge is mostly carried forward from Nehalem. It is a 256KB, 8-way associative design with a 12 cycle load-to-use latency (versus 10 cycles in Nehalem). The L2 has a non-exclusive/non-inclusive relationship to the L1 caches and is a write-back design. The bandwidth and outstanding misses for the L2 cache are the same as for Nehalem – delivering 32B/cycle, with 16 outstanding misses.

L3 Cache and Ring Interconnect

Since Nehalem, Intel is pursuing a more modular, system-on-a-chip like philosophy, where products may share the same core, but have an entirely different system infrastructure. In the past, this differentiation was achieved with a combination of changes to the CPU silicon and the discrete chipset. This relationship always existed between the mainstream x86 cores and the high-end Xeon designs (e.g. Penryn and Dunnington or Nehalem-EP and Nehalem-EX), which feature far more cache, coherency bandwidth, and reliability features. Now that the chipset is firmly integrated with the CPU for all products, this practice now spans Intel's entire mainstream x86 product lines. For each product, Intel will optimize the system architecture for the right balance of performance, power, reliability and cost.
Nearly every aspect of the Sandy Bridge microarchitecture has been redesigned to improve per-core performance and power efficiency. In tandem with these changes to the processor core, the overall system design for products based on Sandy Bridge was rearchitected. The major blocks in Sandy Bridge correspond to three power and frequency domains: the cores and last level cache, the graphics and the system agent. The first two domains are variable voltage and frequency, while the system agent runs at a fixed frequency. The graphics integration in Sandy Bridge is complicated enough to merit a separate entire article. Leaving graphics aside for now, the chip level integration features relevant to the Sandy Bridge microarchitecture include the last level cache (the L3) and system agent, including power management and the display engine.
Sandy Bridge is tied together with a high bandwidth coherent interconnect that spans the three major domains. Nehalem and Westmere used crossbar interconnects, which are extremely efficient and high bandwidth for a small numbers of agents – but must be redesigned to vary the number of

agents. In contrast, Nehalem-EX and Westmere-EX both rely on a ring topology where the wiring and design effort scales better with the number of agents.

Sandy Bridge also employs a ring interconnect between the cores, graphics, L3 cache and system agent (including the display/media engine). The coherent ring is composed of four different rings: request, snoop, acknowledge and a 32B wide data ring. Together these four rings are responsible for a distributed communication protocol that enforces coherency and ordering. The protocol is an enhanced version of QPI with some specific additional features. The rings are fully pipelined and run at the core clock and voltage and bandwidth scales with each additional agent. The scaling is not necessarily perfect though, because of the topology. As messages travel across the ring, they can block access to other agents – reducing the available bandwidth as the average hop count increases. Rings can have very natural ordering properties, which makes design and validation simpler than a more connected topology or on-die packet routing, while preserving many advantages (short wiring runs). It is also easier to reconfigure the number of agents on a ring for different product variants. Each agent on the Sandy Bridge ring includes an interface block that is responsible for the distributed access arbitration. The protocol indicates to each interface one cycle in advance whether the next scheduling slot for the ring is available – and the interfaces simply use the next available scheduling slot for any communications.

The Sandy Bridge ring has six agents – the four cores and cache slices share interfaces, and there is a separate interface for the graphics and another for the system agent. Like Nehalem-EX, the rings are routed in upper metal layers above the L3 cache to reduce area impact.

The L3 cache for Sandy Bridge is shared by the cores, the integrated GPU and the display and media controller. Each of these agents accesses the L3 cache via the ring. It is no longer a single unified entity as in Nehalem and Westmere, but is instead distributed and partitioned for higher bandwidth and associativity (similar to Nehalem-EX and Westmere-EX). There is one slice of the L3 cache for each core, and each slice can provide half a cache line (32B) to the data ring per cycle. All physical addresses are distributed across the cache slices with a single hash function. Partitioning data between the cache slices simplifies coherency, increases the available bandwidth and reduces hot spots and contention for cache addresses.

The ring interface block for each slice of the cache contains an independent cache controller. The cache controller is responsible for servicing requests for the mapped physical addresses and enforcing coherency and ordering. Like Nehalem, the L3 cache is inclusive and the cache controller maintains a set of ‘core valid bits’ that act as a snoop filter (there is also a valid bit for the GPU now). The cache controller also communicates with the system agent to service L3 cache misses, inter-chip snoop traffic and uncacheable accesses.

The L3 cache for Sandy Bridge is scalable and implementation specific. The write-back L3 cache for high-end client models is 8MB and each 2MB slice is 16-way associative. The load-to-use latency for the L3 cache was reduced from 35-40 cycles in Nehalem down to 26-31 cycles in Sandy Bridge (latency varies slightly with the number of hops on the ring).

The latency decreased due to several factors. First, each slice of the L3 is much smaller than Nehalem's 8MB L3 cache, so the latency to access the tags and data arrays has decreased. Second, the ring and L3 now reside in the same clock and voltage domain as the cores (and the core clock is certainly faster than the uncore clock in Nehalem). There is a latency penalty for signals crossing to a new voltage and clock domain. This penalty is determined by the ratio between the two frequencies and can be several cycles. Placing the caches and ring in the same domain avoid this problem. The latency of the ring will increase as more agents are attached; each hop on the ring takes 1 cycle, so the latency actually depends on the relative position of the requesting core and the receiving cache slice.

The first Sandy Bridge implementations have four cores and four cache slices. At 3.4GHz, this translates into a theoretical 435.2GB/s of bandwidth from the last level cache, if all cores are accessing the nearest bank. The bandwidth scales up or down with the frequency and number of cores. Smaller slices (e.g. for low-end parts) are created by eliminating or disabling ways of the cache (at 4-way or 512KB granularity).

The L3 cache is allocated between three different domains (CPU, graphics and non-coherent) with way-level granularity, and ways can be shared between multiple domains. The cache controllers monitor various events and are responsible for maintaining quality of service. To reduce the overhead of sharing, each of the three domains has separate coherency and consistency semantics. The CPU domain uses the familiar x86 model. The graphics domain is semi-coherent and controlled by the drivers; data is flushed into the coherent CPU domain by synchronization instructions. The non-coherent domain is also used by the graphics drivers for certain accesses. In this way, the performance and power overhead for coherency is only imposed when it is needed.

System Agent

The Sandy Bridge system agent contains everything outside the CPU cores, L3 cache and graphics. In previous generations, it was described as the 'uncore', which does little to indicate the function and also seems to have a slightly negative connotation. AMD's terminology for previous designs was the 'northbridge', while IBM calls it the 'nest'. Hopefully, the terminology will synchronize to avoid the sort of rampant confusion caused by linguistic differences.

The system agent for client variants of Sandy Bridge incorporates the memory controller, power control unit (PCU), PCI-Express 2.0, DMI and the display engine. The system agent sits on a fixed voltage and frequency plane and is connected to the ring interconnect, giving it a high bandwidth interface to the rest of the system. Rather than exhaustively describe the system agent, we will focus on the PCU, with a brief mention of the display engine.

One of the most important parts of the system agent is the Power Control Unit (PCU). The PCU is a patchable microcontroller responsible for chip-wide power and thermal management. That encompasses "Turbo-mode", the dynamic voltage and frequency scaling (DVFS) for the

cores/caches and graphics, as well as the memory and I/O interfaces. Nehalem features an aggressive DVFS that iteratively measures voltage, power consumption, temperature and other environmental factors. These measurements are used to adjust the voltage and frequency for the cores, using available thermal and power headroom to increase frequency and single threaded performance. In conjunction, Nehalem uses PFET based power gates to eliminate leakage for idle cores, thereby freeing up more headroom for frequency increases.

The DVFS for Sandy Bridge is more aggressive than its predecessor and shares some characteristics with Moorestown. In reality, the temperature of a chip, heat spreader and the heat sink lags behind the power dissipated – when a chip goes from idle to fully active, temperature rises over a period of time, rather than instantaneously. This is critical, since a cold heat sink actually absorbs more power than a warm one and leakage also increases with junction temperature.

The Sandy Bridge PCU uses a more sophisticated model of the chip that factors in this dynamic thermal capacitance, rather than assuming an instant temperature change (as the PCU for Nehalem and Westmere does). During this period of time where the heat sink temperature is low but rising, it will absorb heat faster from the chip. Sandy Bridge can take advantage of the extra heat transfer to safely dissipate more power than the sustained TDP limit. Depending on the system conditions, Sandy Bridge can exceed the TDP for up to 25 seconds before falling back to a sustainable power level. This capability is particularly useful for workloads with highly varied power usage, as found in client systems. It will also undoubtedly be exploited by savvy overclockers with exotic heat sinks, although this may require special BIOS support to extend the 25 second window. It is very likely that Moorestown uses similar algorithms, given that it seems to have a similar ability to exceed TDP briefly and is exclusively focused on workloads where the chip is idle most of the time, interspersed with short activity bursts.

This new capability relies on precise thermal modeling algorithms in the PCU. These more refined thermal estimates also enhance the number of extra frequency bins available with multiple active cores. In previous products, the frequency limits for a single active core were often much higher than scenarios with two or four cores active. For example, the Core i5 750 has a base clock of 2.66GHz and can reach 2.8GHz (+5%) with 3-4 cores or 3.2GHz (+20%) with 1-2 cores active. While the actual binning for Turbo-mode is a product level decision, Sandy Bridge should improve flexibility for some models.

Previously, the GPU was on a separate die and was power managed by the driver. Since Sandy Bridge integrates graphics and CPUs into a single chip, the PCU can flexibly manage the power and thermal budget between the two components with much higher precision and lower latency. Sandy Bridge can provide higher performance for most applications, by sharing thermal and power resources between the graphics and CPUs, rather than statically allocating a power and thermal budget. For example, when running a CPU-intensive workload where graphics is lightly used, the PCU can allocate more power for the CPU cores, which then translates into higher frequency and performance. In high-end products, reportedly each core can reach 3.8GHz, while the GPU can hit 1.35GHz.

Each core in Sandy Bridge can be power gated off. Since the graphics has its own power plane, it can be controlled with traditional techniques, rather than using on-die power gating. The cache and ring interconnect cannot be shut down or power gated, since they are shared by all components. However, Intel's L3 caches are designed with sleep transistors and other techniques to reduce active and idle power. It is also likely there are ways to reduce the idle power of the ring interconnect when it is not in use, but power gating was not considered an attractive option for Sandy Bridge. Naturally, the system agent must always stay active, since it includes the PCU, and it also receives the base clock over DMI.

The display and media engine in Sandy Bridge has also been substantially reworked. High definition MPEG2, VC1 or AVC streams are decoded in hardware (2 simultaneous streams can be decoded). Unlike previous generations, decoding does not involve the GPU. The fixed function decoding block is more power efficient than the programmable GPU, and also avoids sending data across the chip and waking up the GPU – reportedly achieving a 2X decrease in power consumption for playback. Encoding re-uses many of the decoding fixed function blocks and works with the GPU shader cores through the L3 cache. Demonstrations at IDF suggest a 2X performance gain, although further testing should show a more complete picture. Intel has also release a software development kit so that programmers will be able to take advantage of these new features.

Conclusions and Analysis

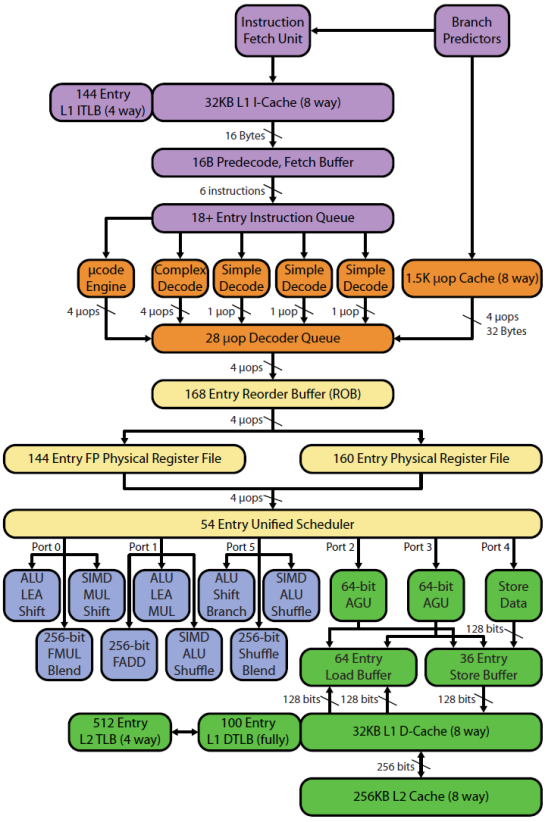
Sandy Bridge is one of the most ambitious and aggressive microprocessors designed at Intel. The degree of complexity and integration is simply astounding. It combines a new CPU microarchitecture, a new graphics microarchitecture, each of which is a substantial departure from the previous generation. On top of that, the chip level integration has taken a huge step forward; with a much more complex system agent and a new L3 cache and ring interconnect shared by all the components. Coherent communication between the CPU and GPU in Sandy Bridge is a substantial advance for the industry and presents many opportunities. Dealing with all these different facets of Sandy Bridge in a single discussion is impossible given the scope of changes.

Sandy Bridge is a fundamentally new microarchitecture for Intel. While it outwardly resembles Nehalem and the P6, it is internally far different. The essence of an out-of-order microarchitecture is tracking, re-ordering, renaming and dynamically scheduling operations to achieve the limit of data flow. Nehalem and Westmere rely on the same mechanisms that date back to the original P6. Sandy Bridge changes the underlying out-of-order engine and uses the more efficient approach taken by the EV6 and P4. That one change alone qualifies Sandy Bridge as a different breed entirely from the P6. But, there are changes in almost every other aspect of the design. The uop cache is a huge improvement for the front-end, largely by eliminating many of the vagaries of x86 fetch and decode. The implementation is quite clever and achieves many of the aims of the P4's trace cache, but in a

far more efficient and reliable manner. AVX improves execution throughput and most importantly, the more flexible memory pipelines benefit almost all workloads.

In the coming year, three new microarchitectures will grace the x86 world. This abundance of new designs is exciting; especially since each one embodies a different philosophy. At the high-end, Sandy Bridge focuses on efficient per-core performance, while Bulldozer explicitly trades away some per-core performance for higher aggregate throughput. AMD's Bobcat takes an entirely different road, emphasizing low-power, but retaining performance. In contrast, Intel's Atom is truly intended to reach the most power sensitive applications. The two high-end microarchitectures, Sandy Bridge and Bulldozer, are shown below in Figure 7. Note that each Bulldozer module would include two integer cores while sharing the front-end and floating point cluster. Also, the floating point cluster in Bulldozer does not directly access memory, instead it uses the memory pipelines in the two attached cores, which then forward results to the FP cluster.

Sandy Bridge



Bulldozer

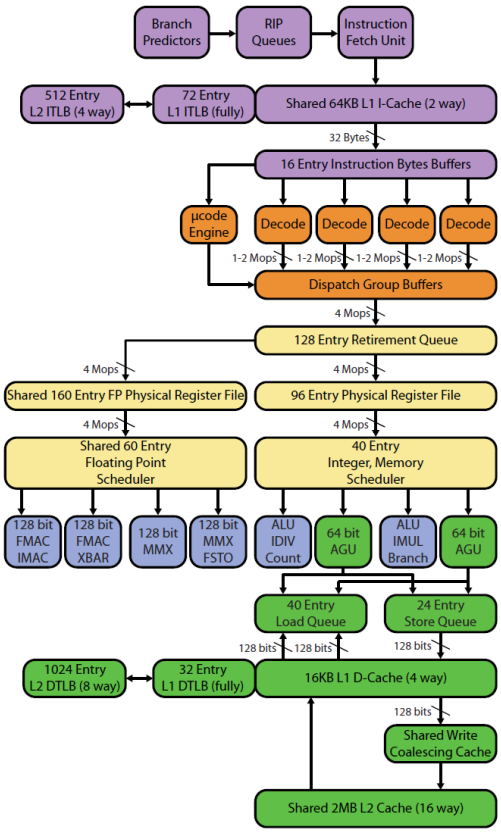


Figure 7 – Sandy Bridge and Bulldozer Microarchitectures

With the limited details, it is hard to predict the chip level performance for products based on these two microarchitectures. Frequencies are still undisclosed, or have yet to be determined and the client and server products will be rather different. In the case of Sandy Bridge, the clock speed should be in the same vicinity as Nehalem or Westmere – however, Bulldozer is clearly intended to run faster, but the frequency will probably be dictated by power consumption. For Bulldozer, there are also numerous details on the integration (e.g. L3 cache design, snoop filter) that are undisclosed. Nonetheless, it is possible to make some educated estimates about the performance of the two microarchitectures.

In looking at the two designs, it is sensible to compare a multi-threaded Sandy Bridge core to a Bulldozer module and separately consider single threaded operation as a special case. Both support two threads although the resources are very different. At a high level, Sandy Bridge shares everything between threads, whereas Bulldozer flexibly shares the front-end and floating point units, while separating the integer cores.

A Sandy Bridge core should have substantially higher performance than a Bulldozer module across the board for single threaded or lightly threaded code. It will also have an additional advantage for floating point workloads that use AVX, (e.g. numerical analysis for finance, engineering). With AVX, each Sandy Bridge core can have up to 2X the FLOP/cycle of a Bulldozer module, although they would be at parity if the code is compiled to use AMD's FMA4 (e.g. via OpenCL). FMA4 will be relatively rare because, while elegant, it is likely to be a historical footnote for x86, supplanted by Intel's FMA3. For software still relying on SSE, the difference between the two should be minimal. In comparison, Bulldozer will favor heavily multi-threaded software. Each module has twice the memory pipelines and slightly more resources (e.g. retirement queue/ROB entries, memory buffers) than a single Sandy Bridge core with two threads, so Bulldozer should do very well in many highly parallel integer workloads that exercise the memory pipelines.

In many ways, the strengths of Sandy Bridge reflect the intentions of the architects. Sandy Bridge is first and foremost a client microprocessor – which requires single threaded performance. Bulldozer is firmly aimed at the server market, where sacrificing single threaded performance for aggregate throughput is an acceptable decision in some cases. Perhaps in future articles, we can examine the components of performance in greater detail (e.g. frequency, IPC, etc.), but for now, high level guidance seems appropriate – given the level of disclosure from both vendors.

Ultimately, we will be waiting for real hardware to see how the Sandy Bridge client performs in the wild. The base clocks, realistic turbo frequencies and power consumption will all be very interesting to observe – and help estimate server performance as well. For now the hardware certainly looks promising and while we await products, we'll have other reports on different aspects of Sandy Bridge to keep us occupied. The design team certainly deserves a round of congratulations for a job well done, redoing the microarchitecture from the ground up while tackling all the integration challenges.