AMD’s Bulldozer Microarchitecture

Trends in Computer Architecture

In many respects, the evolution of mainstream x86 microprocessors over the last 10-15 years has been very consistent, steady and gradual, with a few exceptions. Since the advent of their first out-of-order designs, both AMD and Intel have been using very similar microarchitectures throughout successive generations. The notable exception is Intel’s Pentium 4, which proved to be poorly suited for most of the market. That aside, the heritage of AMD’s designs is very clearly descended from the K7 generation, while Intel’s microarchitectures are derived from the Pentium Pro. AMD and Intel both focused on the same initial design goals: maximal single threaded performance given reasonable manufacturing costs. Around 2003, the additional constraint of power efficiency became essential – both to accommodate affordable cooling solutions, electrical delivery and also the rise of notebooks. While this change (amongst other things) doomed the Pentium 4, the challenges of power efficiency were mostly met by continuous improvements rather than radical departures. The x86 ecosystem is largely ruled by two microarchitectures, Intel’s Nehalem and AMD’s Istanbul (for simplicitly we use Istanbul to refer to all of Family 10h). Together these two designs span the entire PC market from 10W notebooks to 130W servers, with a third for the somewhat different ultra-low power market (Intel’s Atom).

At the same time, computer architecture as a whole has been diversifying quite rapidly. Many of the various RISC architectures have died out (Alpha, PA-RISC) or moved on to different markets (MIPS). Those that remain are pursuing fairly different goals. IBM is focused on maximizing single threaded and aggregate system performance using every resource available with only cursory regard to cost. Oracle nee Sun is following the path of throughput computing, sacrificing single threaded performance to simplify design and achieve power efficiency for low socket count servers. On the other hand, Fujitsu still has a relatively traditional microarchitecture, with mainframe-like emphasis on reliability.

Recently, GPUs have emerged as a possible alternative. The standard bearer of this camp is Nvidia’s Fermi architecture. GPUs are conceptually akin to Sun’s throughput focus, but with programmability and reliability sacrifices. The advantage though is that GPUs currently benefit from high volumes due to the consumer mass market. The high volume aspect is threatened though, by the integration of CPU and GPU being pursued by Intel and AMD.

AMD Diversifies

In this context, AMD has stepped away from their historic design point and re-evaluated their approach to the x86 market. Rather than pursuing one core to address the vast majority of the market, they opted to develop two new microarchitectures, Bobcat and Bulldozer. Bobcat is focused on low cost, low power and GPU integration for notebooks and low-end desktops. Llano is focused

on higher power and performance with GPU integration, but uses a derivative of the existing Istanbul microarchitecture. Like the original K8, Bulldozer is first and foremost aimed at the high-margin server market and may be used in performance sensitive desktop systems as well.
AMD's hope is that this three pronged approach will enable them to more carefully optimize for various customers and differentiate with respect to Intel. In some respects, this is a sign of maturity – AMD is capitalizing on the acquisition of ATI and trying to more carefully address the market and outmaneuver Intel. At the same time, this strategy is fairly risky. Designing and validating a new x86 microarchitecture is quite challenging; this is the first time that AMD has done two simultaneously. Moreover, AMD is concurrently making the shift from an integrated device manufacturer to a fabless company that relies on 3rd party manufacturing from Global Foundries. Despite the risk, it is nice to see more diversity in computer architecture amongst major players.

The Philosophy of Bulldozer

AMD first managed to break into the server market in 2003 with the K8, thanks to the 64-bit extended instruction set and system architecture choices such as an integrated memory controllers and on-die interconnects. Intel certainly helped out as well, since the Pentium 4 was a distinctly unfriendly product for servers. AMD eventually hit an impressive 25% market share, and over 50% share in the lucrative 4-socket server market by 2005/6. The key to AMD's success was that they provided exactly what consumers wanted (x86-64 and good server performance), while Intel was distracted with Itanium and the P4. In essence, they found an area where Intel could not (or would not) focus, and then put all their efforts into addressing customer needs and were able to change the rules of the game.
By mid to late 2006, AMD's server fortunes were on the decline as Intel launched the 65nm Core 2 Duo (a dual core), and a multi-chip package (MCM) that paired together two chips to provide 4 cores in total. AMD's 65nm, 4-core Barcelona (and later derivatives) improved the competitive situation somewhat, but was not enough to really change the overall direction of the market. This reversal accelerated in 2008, when Intel launched the 45nm, 4-core Nehalem, their first server microprocessor to use the so called Quick Path Interconnect – really an integrated memory controller and an on-die coherent interconnects. With that change in system architecture, AMD's last major advantage was gone and Intel had much higher performance at comparable or better power efficiency. AMD's response in mid-2009 was the 45nm, 6-core Istanbul – which did not quite manage to achieve parity. In early 2010, Intel completed their return to server ascendancy with the 32nm, 6-core Westmere and the 45nm, 8-core Nehalem-EX, which targeted 4-socket designs and was the first high-end server part with the Quick Path Interconnect. AMD's failure here was largely a result of trying to match Intel's superior manufacturing and resources head-on.
In the earlier part of 2010, AMD launched Magny-Cours, which pairs together two existing 45nm, 6-core CPUs in a single MCM. For highly threaded and parallel applications, Magny-Cours is efficient

and affordable. For instance, it is very well suited for certain portions of the HPC market. However, applications which favor single threaded performance will inevitably do best running on an Intel-based solution. Unfortunately, differentiating solely based on the number of cores has a cost – die area. From a manufacturing stand point, each Magny-Cours uses a pair of 346mm^2 chips, compared to a single 246mm^2 die for Westmere and 684mm^2 for Nehalem-EX.

Philosophically, Bulldozer seems to learn from the lessons of the past decade. AMD is stepping back from the pursuit of single threaded performance, to emphasize throughput. The cores are not lightweight, as with a GPU or with Niagara; the single threaded performance should actually be higher than the previous generation Magny-Cours and comparable to current Intel designs. However, in determining project goals for Bulldozer single threaded performance was consciously sacrificed to meet what the team determined was a more optimal overall design point. This stands in contrast to Intel, where single threaded performance is still the first and foremost design target for designs like Sandy Bridge. This is acknowledgement that AMD cannot beat Intel on single threaded performance, and it would be a repetition of the last 3 years to attempt such an endeavor. Instead, they are trying to change the rules of the game, by focusing on the core count and highly parallel server workloads.

Bulldozer is the first x86 design to share substantial hardware between multiple cores, in some cases blurring the traditional notion of a core. Current x86 designs share the last level cache, power management and external interfaces such as the memory controllers, coherent interconnects and other I/O between 2-8 cores. Bulldozer is a hierarchical design with sharing at nearly every level. Each module or compute unit (i.e. a pair of cores) share an L1I cache, floating point unit (FPU) and L2 cache, saving area and power to pack in more cores and attain higher throughput – albeit with a slight cost in terms of per-core performance. All modules in a chip share the L3 cache, Hypertransport links and other system components.

While sharing the front-end and FPU seems radical compared to today's x86 designs, it is a natural evolution in the multi-core era. The front-end has to deal with a lot of the complexity of the x86 instruction set, which leads to power hungry decoders and large structures like microcode. Floating point hardware also consumes a great deal of area and power and is rarely utilized over 40% – so sharing between two cores is an excellent way to gain back area and power with a minor performance loss. In many ways, AMD is positioning this high degree of sharing as an alternative to multi-threading, which is used by almost every other high performance CPU; and there is some truth to this claim.

Bulldozer itself is somewhat of a contradiction – it is a substantial departure from the previous generation Istanbul, yet in most parts of the design, it is also clearly descended from AMD's previous work.

Bulldozer is a high frequency optimized CPU, a so called speed demon. This approach has fallen out of popularity in the x86 world, due to Intel's misadventures with the Pentium 4. In all fairness though, many of the Pentium 4's problems were unrelated to high clockspeed and more closely tied to the actual microarchitecture. In the high-end server world, IBM has successfully pursued high clock speeds with the POWER7. So a speed demon approach can work out successfully. Bulldozer has a fairly lengthy pipeline, to minimize the gate delays per stage. AMD was unwilling to share any specifics on gate delays, although some discussions at comp.arch suggest a target of ~17 gate delays vs. ~23 for Istanbul. To tolerate the increased latencies necessary for a high frequency target and to efficiently share resources between cores, Bulldozer introduces decoupling queues between most major stages in the pipeline.

This article describes in detail the architecture and pipeline of the Bulldozer core a 64 bit, 4 issue super-scalar, out-of-order MPU with 48 bit virtual and physical addressing. The first Bulldozer based product, Interlagos, is an 8-core (4 module) design implemented in Global Foundries' high performance 32nm SOI process, using high-K gate dielectrics and metal gate stacks, with a gate first approach. Two chips may be packaged together in a single MCM to achieve as many as 16 cores in a single socket. Bulldozer is compatible with all the latest Intel x86 extensions (SSE 4, AES-NI and AVX), and also the AMD XOP, FMA4 extensions and the Light Weight Profiling specification.

Since Bulldozer is not expected till 2011 (most likely in the second half), AMD was reluctant to share product level details. Instead they focused on describing the core at Hot Chips, and avoided mentioning many details pertaining to other portions of the chip. Even for some features within the core, they were unwilling to disclose substantial details – especially in the front-end.

System Architecture

The socket infrastructure for Magny-Cours was explicitly designed to accommodate future Bulldozer-based CPUs. This includes both the 1944-pin CPU socket itself and also the approach of combining two CPUs (referred to as nodes) in a single package. Inheriting the system architecture from a previous generation is wholly logical and wise decision on AMD's part. First of all, AMD's server volumes are relatively low. The longer a platform lasts, the greater the volume of the platform and thus partners (and AMD itself) can justify a greater investment in the ecosystem. AMD has historically tried to keep their platforms as stable as possible to encourage partner adoption, and even Intel is similarly constrained (primarily at the high-end of the market). Second, Bulldozer is already a very risky project combining a new microarchitecture, a new process technology and a new manufacturing arrangement. Re-using parts of the system architecture avoid more risk and complexity.

As AMD indicated in their presentation at the last Hot Chips, the packaging arrangement for Magny-Cours is intended to achieve the majority of the performance and benefits of a 2N-socket system, but in the foot print of an N-socket system. The draw back is that no more than 4 Magny-Cours can be connected without a node controller. In contrast, Intel's Nehalem-EX can gluelessly scale up to 8 sockets.

Interlagos, the first Bulldozer instantiation is an 8-core device. However, since it is intended to fully inherit the socket infrastructure of the previous generation, the first products will offer up to 16 cores using an MCM. Figure 1 below shows the expected system architecture for a single Interlagos node (i.e. one die in an MCM) and the actual system architecture for a Magny-Cours MCM, Westmere-EP and Nehalem-EX. Figure 1 readily demonstrates the hierarchy in Interlagos (e.g. shared L2 caches) compared to the relatively flat topology of Intel's offerings and the previous generation Magny-Cours. In AMD's terminology, each pair of cores (which share a front-end, FPU and L2 cache) is referred to as a module, or a compute unit. Note that the cache hierarchies in Intel and AMD designs are quite different. Intel's CPUs use an inclusive last level cache to act as a snoop filter, so Westmere can only cache 12MB of data total. AMD favors victim caches, which are mostly exclusively so that separate data may be held in the L2 and L3 caches. The theoretical cache capacity of an Interlagos node is estimated to be 16MB, while each node in a Magny-Cours MCM can hold up to ~9MB; in reality data will be replicated between caches within a node, and also within a single MCM, reducing the effective capacity. The overall benefits of Bulldozer's throughput focus seem clear – Intel's roadmaps for 2011 mostly call for 8 and 10 core products, possibly leaving an opportunity for AMD.

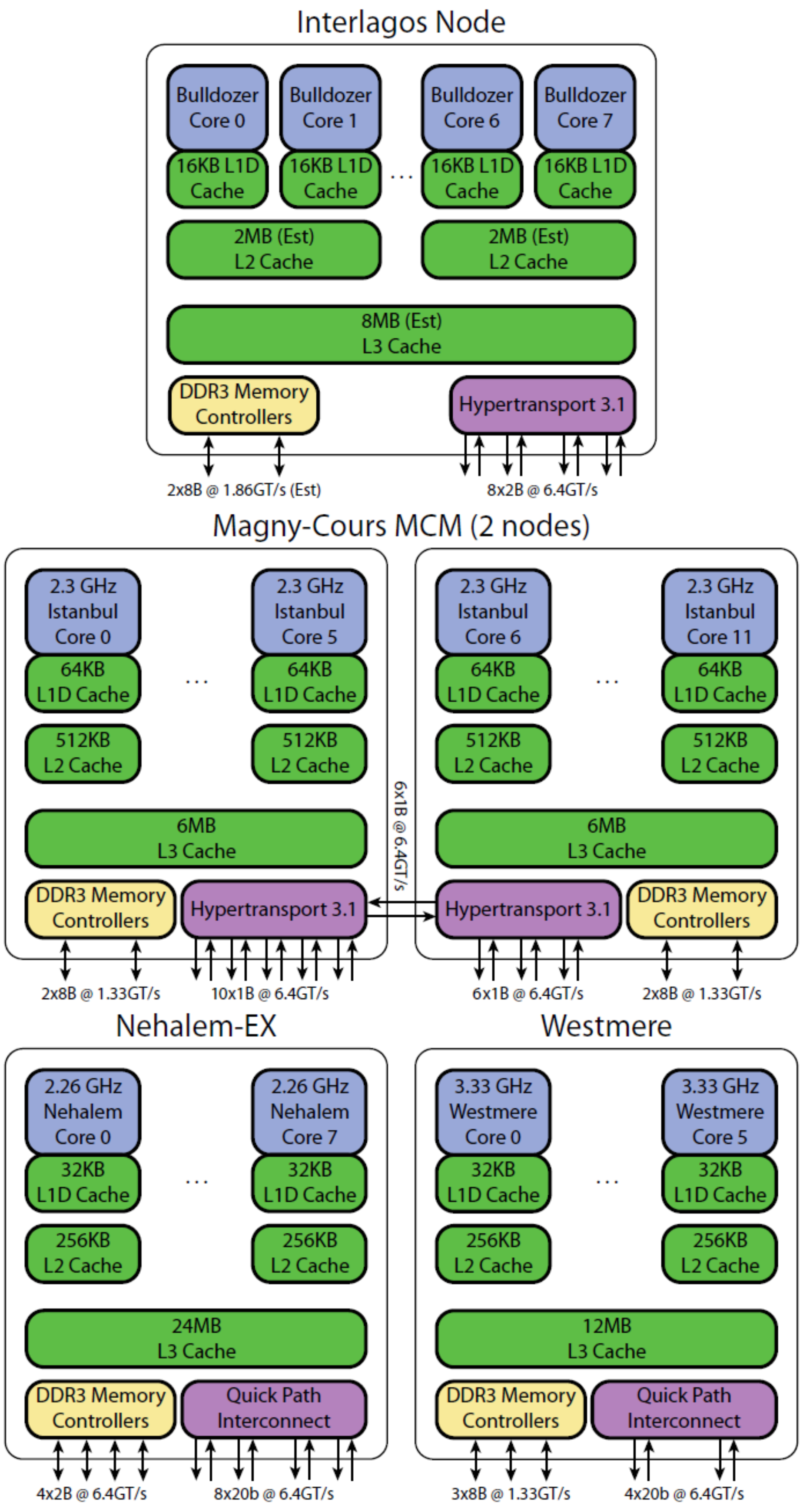

Interlagos Node
Bulldozer Core 0
Bulldozer Core 1
Bulldozer Core 6
Bulldozer Core 7
16KB L1D Cache
16KB L1D Cache
...
16KB L1D Cache
16KB L1D Cache
2MB (Est) L2 Cache
2MB (Est) L2 Cache
8MB (Est) L3 Cache
DDR3 Memory Controllers
Hypertransport 3.1
2x8B @ 1.86GT/s (Est)
8x2B @ 6.4GT/s
Magny-Cours MCM (2 nodes)
2.3 GHz Istanbul Core 0
...
2.3 GHz Istanbul Core 5
64KB L1D Cache
64KB L1D Cache
512KB L2 Cache
512KB L2 Cache
6MB L3 Cache
DDR3 Memory Controllers
Hypertransport 3.1
2.3 GHz Istanbul Core 6
...
2.3 GHz Istanbul Core 11
64KB L1D Cache
64KB L1D Cache
512KB L2 Cache
512KB L2 Cache
6MB L3 Cache
Hypertransport 3.1
DDR3 Memory Controllers
6x1B @ 6.4GT/s
2x8B @ 1.33GT/s
10x1B @ 6.4GT/s
6x1B @ 6.4GT/s
2x8B @ 1.33GT/s
Nehalem-EX
2.26 GHz Nehalem Core 0
...
2.26 GHz Nehalem Core 7
32KB L1D Cache
32KB L1D Cache
256KB L2 Cache
256KB L2 Cache
24MB L3 Cache
DDR3 Memory Controllers
Quick Path Interconnect
4x2B @ 6.4GT/s
8x20b @ 6.4GT/s
Westmere
3.33 GHz Westmere Core 0
...
3.33 GHz Westmere Core 5
32KB L1D Cache
32KB L1D Cache
256KB L2 Cache
256KB L2 Cache
12MB L3 Cache
DDR3 Memory Controllers
Quick Path Interconnect
3x8B @ 1.33GT/s
4x20b @ 6.4GT/s

Figure 1 – Bulldozer System Architecture

Each Interlagos die contains two 64-bit DDR3 memory controllers. Magny-Cours was limited to operation at 1.33GT/s, or 21.3GB/s of memory bandwidth, which is the minimum expected for Interlagos. It is quite possible, even likely, that AMD will support faster operation, as there are JEDEC compliant DDR3 variants that operate at 1.86GT/s, although the fastest 2.13GT/s may be out of reach for servers. The four 16-bit wide coherent HyperTransport (HT) interfaces run at 6.4GT/s or 102GB/s of raw coherency traffic. However, ccHT has a variable length CRC algorithm for packets, thus the usable bandwidth is lower (for comparison, Intel's fixed length CRC uses 20% of the raw bandwidth).

In AMD's MCM, two die are paired together and connected using a full width ccHT lane and a second half-width lane. Only one of the two dice can connect to external non-coherent I/O, but both devices will connect to other CPUs in the system. In aggregate, the MCM has an impressive 42.6GB/s of memory bandwidth and 102GB/s of raw HT bandwidth – equally dividing both between the two dice. It is likely that Interlagos will have even more memory bandwidth, although how much more is hard to say.

Even though the first Bulldozer implementations are somewhat constrained by socket compatibility, there are still a number of opportunities for improvement. For instance, the coherency protocol is still undisclosed and has plenty of room for improvement. As a more concrete example, consider the snoop filter. For Magny-Cours, 1MB out of the 6MB L3 cache was repurposed for a snoop filter that tracks data from local memory that is cached elsewhere in the system. Each 64B cache line holds 16 snoop filter entries in a 4-way set associative arrangement, for a total of 256K entries that index 16MB of cache. With the snoop filter enabled, each Magny-Cours contains 8MB (or 128K lines) of usable cache (3MB L2 + 5MB L3), thus a 4-socket system could in theory have up to 64MB of cached data, although for most situations, there is substantial replication between caches.

Since Interlagos is on 32nm, rather than 45nm, it is natural to expect more cache. Interlagos probably has 16MB of cache per die, roughly double Magny-Cours. Assuming this is true, a 2MB snoop filter would be a reasonable starting point. Alternatively, AMD may have taken an entirely different approach, such as a full directory based cache coherency protocol (similar to Itanium).

Shared Instruction Fetch

Sharing between cores is a key element of Bulldozer's architecture, and it starts with the front end. The front-end has been entirely overhauled and is now responsible for feeding both cores within a module. Bulldozer's front end includes branch prediction, instruction fetching, instruction decoding and macro-op dispatch. These stages are effectively multi-threaded with single cycle switching between threads. The arbitration between the two cores is determined by a number of factors including fairness, pipeline occupancy and stalling events. Each of these major stages is decoupled from the next, by an appropriate queue or pair of queues. The front-end for Bulldozer is shown below in Figure 2.

Since Bulldozer is a high frequency design, branch prediction is critically important. Deeper pipelines tend to have more instruction in-flight at a given time, thus mispredicts can result in squashing more instructions. The number of instructions squashed is directly related to the wasted energy and performance. Historically speaking, Intel has invested far more resources and expertise in branch prediction and as a result has the most advanced and highest performance predictors. While Bulldozer is a substantial improvement over Istanbul, the end results are hard to assess without seeing the full details or a testable product.

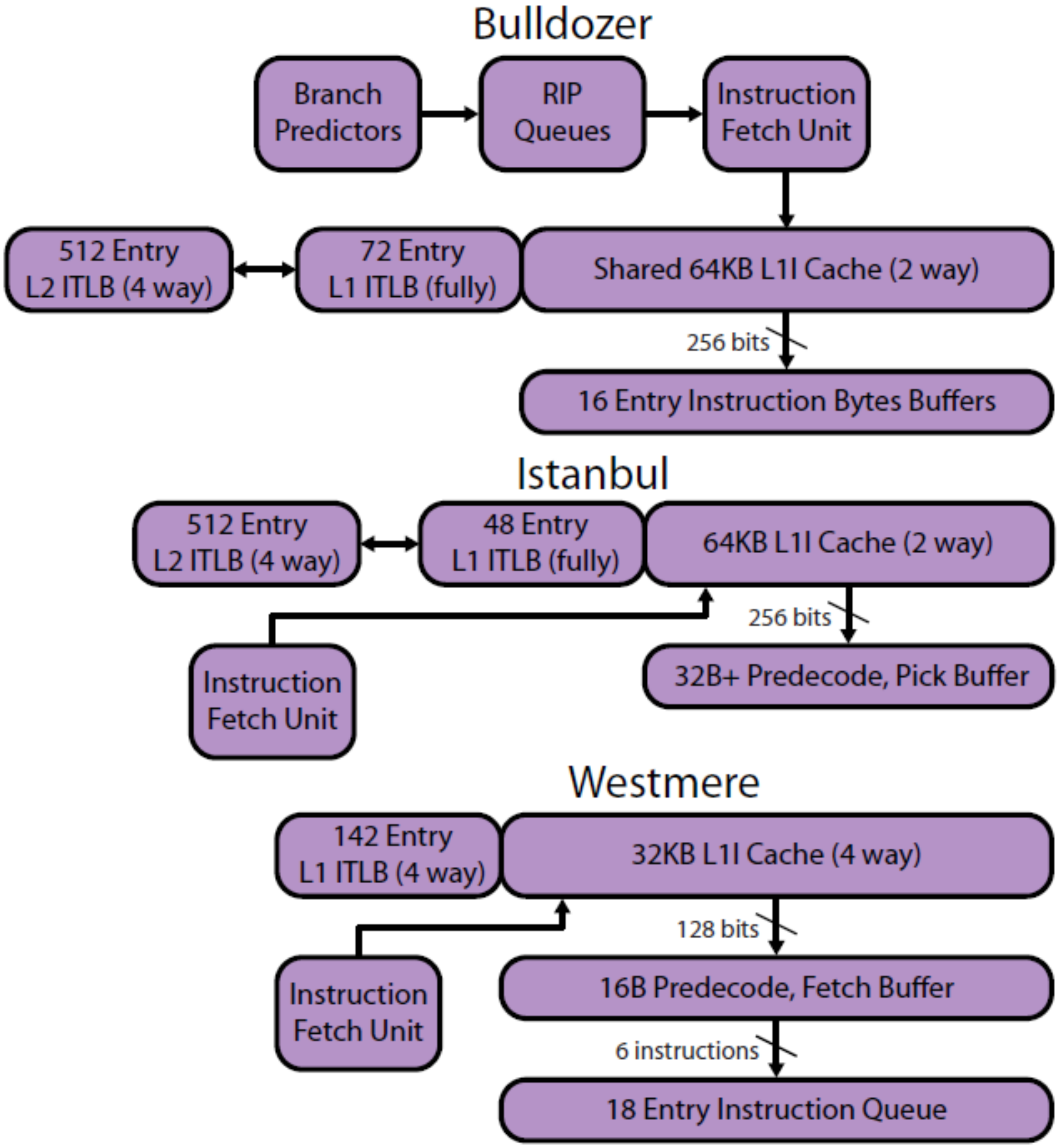


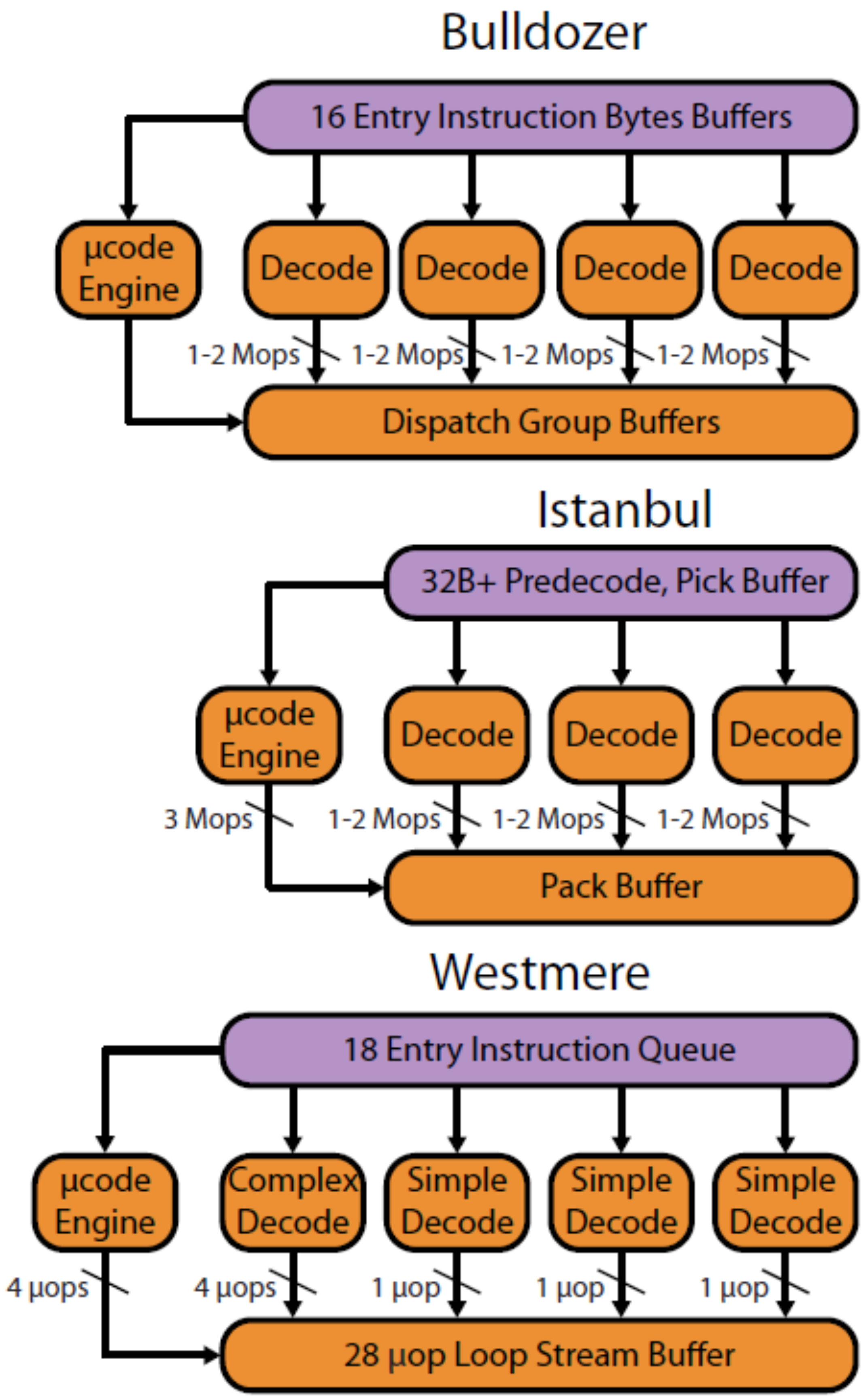
Figure 2 – Bulldozer Instruction Fetch and Comparison

The branch predictor is shared by the two cores in each module and decoupled from the instruction fetching via a pair of prediction queues (one queue per core). The branch predictor can run-ahead and will continue to predict new relative instruction pointers (RIPs) unless the queues are full.
The first step in branch prediction is determining the direction – whether a branch is taken or not. AMD previously used a local predictor, a global predictor and a selector that would choose which of the two predictors to use. However, they were extremely coy about the predictors used in Bulldozer, other than to indicate that they did not use multi-level predictors. It is possible that AMD included a loop detector, something Intel introduced in the Pentium M.
Once a branch is predicted as taken, the next step is to determine the target. For branches with a single target address, the branch target buffer (BTB) has been substantially expanded and now uses a two level hierarchy, similar to Nehalem. The L1 BTB is a 512 entry, 4-way associative structure that resolves predictions with a single cycle penalty to the pipeline. The L2 BTB is much larger with 5120 entries and 5-way associativity, but the extra capacity will cost additional latency for a L2 BTB hit. The BTBs in Bulldozer are competitively shared by both cores, but provide greater coverage than the 2K entry BTB in Istanbul.
The other key structures used to predict the target address of a branch were not described in detail by AMD. They confirmed that for indirect branches (those with more than one target address), Bulldozer also includes a 512-entry indirect target array. Istanbul included a 512 entry indirect array; it is possible that this was kept the same size, although it would make sense to increase the number of entries to account for both cores. Bulldozer includes the familiar call/return stack, which is replicated per thread, rather than shared. Istanbul's 24 entry return address stack could be corrupted by a branch misprediction, which would result in subsequent returns being mispredicted. Bulldozer has mechanisms to repair the return stack, avoiding this corruption issue and decreasing return mispredictions; a feature first seen in Nehalem.
The branch prediction and the RIP queue can effectively run ahead of the instruction fetch unit in Bulldozer. This helps the two cores smoothly share the branch prediction hardware and tolerate longer latencies in the front-end. Just as importantly, by having multiple RIPs ready at a given point in time, the fetch unit can prefetch the instruction stream for branches in the BTBs and indirect array. This prefetching hides some of the fetch latency and enables greater memory level parallelism for the instruction caches.
Once a RIP is placed into the prediction queue and passed to the next stage, the fetch unit accesses the dynamically shared ITLBs and L1 I-cache. The L1 ITLB is fully associative with 72 entries for the various page sizes. Istanbul did not have 1GB pages for the ITLBs, and it is almost certain this has been rectified for Bulldozer. The break down of the 72 entries in the L1 ITLB has not been disclosed, but judging by design choices made in prior generations, over half the entries will be for 4KB pages, with a lesser number for larger (2MB or 1GB) pages. The backing L2 ITLB which only holds 4KB pages has been expanded, with 512 entries and 4-way associativity.
The L1 instruction cache should be very familiar, as it has the same organization as Istanbul's – 64KB and 2-way associative and probably contains similar pre-decode information. What is most

puzzling about the L1I is the low associativity – essentially one way for each of the two cores sharing the instruction cache.While each cache line is 64B, the fetcher retrieves 32B of instructions each cycle into the Instruction Byte Buffers (IBB), taking two cycles to complete a fetch.
The IBB is the last stop before decoding and acts as the decoupling queue between fetch and decode. Accordingly, there are two IBBs, one dedicated per core. Each IBB contains 16 entries, sometimes called dispatch windows. Each window holds 16B of x86 instructions, thus the total IBB capacity is 256B per core.

Shared Instruction Decode

Before delving further into decoding, it is useful to discuss some terminology. Intel refers to the variable length x86 instructions as macro-operations. These can be quite complex, with multiple memory and arithmetic operations. Intel refers to their own simpler internal, fixed length operations as micro-ops or uops. AMD's terminology is subtly (and confusingly) different. In AMD parlance, x86 instructions are referred to as AMD64 instructions – again, variable length and potentially quite complex. An AMD macro-operation is an internal, fixed length operation that may include both an arithmetic operation and a memory operation (e.g. a single macro-op may be a read-modify-write). In some cases, AMD also refers to these macro-ops as complex ops or cops. An AMD uop is an even simpler, fixed length operation that performs a single operation: arithmetic, load or store – but only one, and not a combination. So in theory, an AMD macro-op could translate into 3 AMD uops. The best way to think about AMD's arrangement is that macro-ops are how the out-of-order microarchitecture tracks work, while uops are executed in execution units. For the rest of this article, we will endeavor to use AMD's terminology.
As with Istanbul, Bulldozer classifies instructions into three categories: FastPath Singles, which emit a single macro-op, FastPath Doubles which emit two macro-ops and Microcode or VectorPath (everything else). Since AMD's macro-ops are fairly powerful and complex, most instructions tend to decode to a single macro-op. However, that is not likely the case for 256-bit AVX instructions since Bulldozer's execution hardware is 128-bit wide. A 256-bit AVX instruction could actually generate two 128-bit loads and two 128-bit floating point operations. When faced with a similar situation for 128-bit SSE and the K8, AMD took the approach of mapping 128-bit SSE to two macro-ops. It is a good guess (although not certain) that Bulldozer also decodes 256-bit AVX instructions into two macro-ops, which has implications for the resources used in the pipeline from renaming to execution.



Bulldozer
16 Entry Instruction Bytes Buffers
µcode Engine
Decode
Decode
Decode
Decode
1-2 Mops
1-2 Mops
1-2 Mops
1-2 Mops
Dispatch Group Buffers
Istanbul
32B+ Predecode, Pick Buffer
µcode Engine
Decode
Decode
Decode
3 Mops
1-2 Mops
1-2 Mops
1-2 Mops
Pack Buffer
Westmere
18 Entry Instruction Queue
µcode Engine
Complex Decode
Simple Decode
Simple Decode
Simple Decode
4 µops
4 µops
1 µop
1 µop
1 µop
28 µop Loop Stream Buffer

Figure 3 – Bulldozer Decode and Comparison

The decode phase for Bulldozer, shown in Figure 3, has been improved, but the changes are far less dramatic than for fetching. The decoding begins by inspecting the first two of the 16B windows in the IBB for a single core. In many circumstances, instructions can be taken from both windows, but there are restrictions based upon alignment, number of loads and stores, branches, and other factors which can restrict decoding to a single 16B window.

To accommodate both cores, Bulldozer's decode stage has been widened. Bulldozer can decode up to 4 instructions per cycle. After examining the instruction window, the decoders translate each x86 instruction into 1 or 2 macro-operations and place them into a queue for dispatching. Microcoded instructions (i.e. those requiring more than 2 macro-operations) are handled by the microcode ROM and probably cannot proceed in parallel with FastPath instructions. While AMD did not disclose how the microcode works, they did imply that the microcode will at least maintain the same performance as in prior generations (i.e. emitting at least 3 macro-operations per cycle). Given that it is shared between two four-issue cores, they may have modestly improved the performance to emit 4 macro-ops per cycle.

In addition to having an extra decoder, Bulldozer is the first AMD CPU with branch fusion, whereby an adjacent arithmetic test or comparison and a jump instruction are decoded into a single macro-op. There are some restrictions on branch fusion, but they are likely to be relaxed over time. For example, Intel was the first to introduce this feature in the Core 2 Duo, but only for 32-bit mode. Subsequent iterations generalized the feature to a greater degree (64-bit mode and more combinations of arithmetic and control flow). Bulldozer also has the side-band stack optimizer, which renames the stack pointer and thus breaks dependencies between instructions that implicitly reference the stack pointer.

Once the instructions have been decoded into macro-ops, they are placed into a queue where they are put into dispatch groups of up to four macro-ops and then sent to one of the two cores.

Integer Cores

Like all previous designs from AMD (and in contrast to Intel), Bulldozer separates the integer and floating point schedulers, register files and execution units. In proof that Sutherland's Wheel of Reincarnation applies to more than just graphics, Bulldozer employs a co-processor model for floating point and SIMD execution that is shared by both cores in a module – reminiscent of the days when x87 floating point co-processors would reside on a separate chip altogether. One advantage of this more formalized separation is that the floating point cluster might eventually be replaced or supplemented by a GPU shader array, an evolution of Bulldozer to fit the 'Fusion' mold. This co-processor model is an example of a substantial change that is also familiar from previous AMD CPUs, the resemblance is clear from Figure 4 below.

Each cycle, a group of up to four macro-ops is dispatched to one of the dedicated cores. The macro-ops are allocated into the 128 entry retirement queue, which is responsible for maintaining the bookkeeping logic for each macro-op in flight. Memory operations must also allocate an entry in the appropriate load or store queue, to maintain x86 consistency. Within each dispatch group, any integer and memory macro-ops are renamed into the 96-entry physical register file (which contains both architectural and speculative registers). However, any FP or SIMD macro-ops will be sent to the floating point cluster to continue execution, although the retirement status is tracked in the integer core. Note that a FP or SIMD memory access will dispatch both a memory macro-operation to the integer core and an execution macro-operation to the FP cluster.

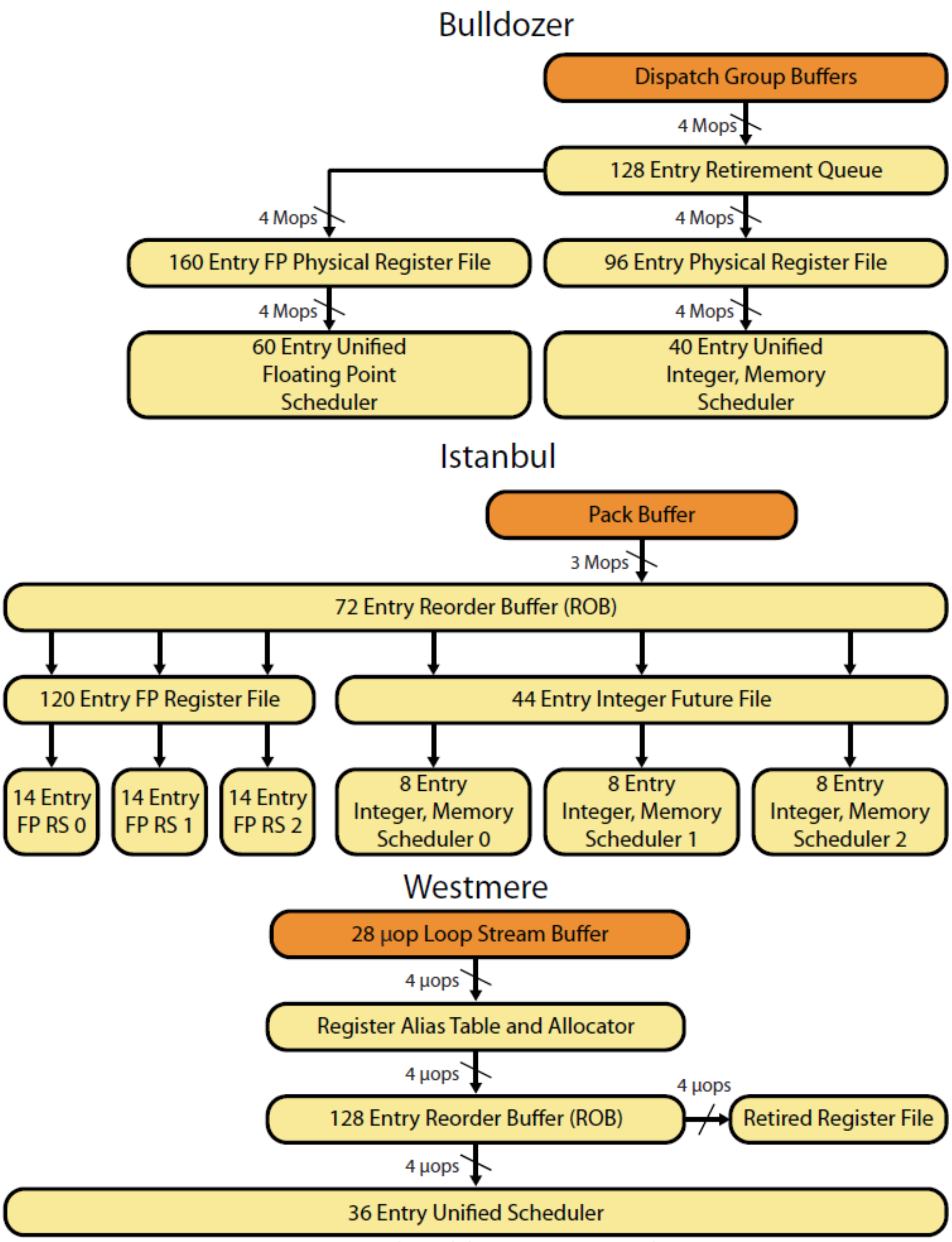

Bulldozer
Dispatch Group Buffers
4 Mops
128 Entry Retirement Queue
4 Mops
4 Mops
160 Entry FP Physical Register File
96 Entry Physical Register File
4 Mops
4 Mops
60 Entry Unified Floating Point Scheduler
40 Entry Unified Integer, Memory Scheduler
Istanbul
Pack Buffer
3 Mops
72 Entry Reorder Buffer (ROB)
120 Entry FP Register File
44 Entry Integer Future File
14 Entry FP RS 0
14 Entry FP RS 1
14 Entry FP RS 2
8 Entry Integer, Memory Scheduler 0
8 Entry Integer, Memory Scheduler 1
8 Entry Integer, Memory Scheduler 2
Westmere
28 µop Loop Stream Buffer
4 µops
Register Alias Table and Allocator
4 µops
4 µops
128 Entry Reorder Buffer (ROB)
Retired Register File
4 µops
36 Entry Unified Scheduler

Figure 4 – Bulldozer Out-of-Order Engine and Comparison

Bulldozer’s physical register file (PRF) based renaming is a fundamental change to the out-of-order microarchitecture. In some previous out-of-order designs (e.g. integer renaming in K7 derivatives up to Istanbul, P6 derivatives up to Nehalem) state was tracked in two separate structures – a re-order buffer (ROB) or future file and an architectural register file. The ROB is implemented as a single structure that contains both the data value for each renamed register and also status information for each in-flight macro-op. The architectural register file contains the data value for each architectural register (e.g. RCX or XMM0). Once a macro-op retires, its renamed register value is written into the appropriate architectural register. So in a ROB based design, the state is held into two structures, one for speculative state and one for architectural state.

In microarchitectures using a physical register file (such as Bulldozer, IBM’s Power4, Power5 and Power7, DEC’s 21264 and the Pentium 4), there also are two structures to hold the state, but they are divided based on function. One structure is the physical register file, which holds all the data values – both for speculative renamed registers and architectural registers. The other structure is what AMD calls the retirement queue, which holds a pointer to the appropriate entry in the PRF and also status information about each in-flight macro-op and the associated register (e.g. is the register speculative or architectural). This microarchitecture enables lower power retirement and rollback, by manipulating the retirement queue pointers that map into the PRF, rather than copying register values. This is the fundamental difference between the two approaches: ROBs keep data with the status information, while a PRF separates the data from the status information. In Bulldozer, up to 4 macro-ops can be retired each cycle, matching the throughput in the rest of the CPU. Branch mispredicts are handled by a flash clear, which likely reverts the retirement queue to a prior known good state.

Once renamed, macro-operations are placed into the 40 entry unified scheduler where they are held until all the necessary resources, such as source operands are available. When uops are ready, the scheduler will issue up to four of the oldest uops to the appropriate execution units.

Shared Floating Point Cluster

The floating point cluster for Bulldozer has its own four-wide out-of-order execution facilities including renaming, scheduling and register files. As previously noted though, it relies upon the integer cores for handling any loads and stores and also retiring macro-ops.

When a dispatch group is sent to a core, any FP or SIMD macro-ops are allocated an entry in the retirement queue, just like the integer macro-ops. However, the floating point or SIMD macro-ops are then sent to the FP cluster for renaming, scheduling and execution.

The incoming FP or SIMD macro-operations follow a very similar flow to the integer side. The FP and SIMD macro-ops are renamed into a physical register file (a similar design to the integer PRF). The PRF is dynamically shared between the two cores, and contains 160 entries. AMD has not disclosed the width of these PRF entries. If the PRF entries are 128-bits wide, then each 256-bit instruction will probably decode as two macro-ops and use two entries in the PRF, scheduler and the

retirement queue, reducing the effective instruction window. This seems to be the most probable scenario, although there is a chance that the FP cluster has 256-bit registers.
Once macro-ops are renamed, they are moved into the scheduler. The unified FP scheduler contains 60 macro-operations. When the inputs for a uop are ready (either present in the scheduler or imminently available through the forwarding network), the uops are issued to the appropriate execution pipeline – which is described in the next section. Once a macro-operation is completed, it will signal to the appropriate core for retirement.
The first part of the floating point pipeline is driven by the dispatch logic, which is multithreaded, with switching at a single cycle granularity. Once macro-ops have been placed into the scheduler, there is no longer a distinction between macro-ops from the two cores. Thus the scheduler and execution units are effectively simultaneously multithreaded.

Integer Execution Units

The Istanbul microarchitecture was conceptually oriented around a set of three lanes. Each lane had a dedicated scheduler, tied to largely identical group of integer functional units. Bulldozer abandons the idea of lanes with dedicated schedulers, in favor of a more flexible, unified 40 entry scheduler that can issue to any of the execution units. As shown in Figure 5, Bulldozer features a substantially different assortment of functional units. In Istanbul, the three lanes had a full ALU and an AGU to simplify scheduling by making each lane identical. With Bulldozer, this is no longer necessary. Four ALUs and AGUs would be very power and area inefficient, while providing little additional benefit. Thus to improve throughput (by decreasing core size), AMD reduced the number of integer execution units.
Bulldozer’s integer execution units can be thought of as two mostly identical groups (0 and 1), where each group is composed of an AGU and an ALU. The two AGUs (AGU 0 and 1) are identical and perform the address calculations that feed into the load-store unit and cache hierarchy. The integer execution units (EX 0 and EX 1) are each a fully featured ALU capable of executing the vast majority of integer operations. There are some slight differences between the two pipelines though. EX 0 is responsible for POPCNT and LZCOUNT operations and also contains a variable latency, unpipelined integer divider. EX 1 has a pipelined integer multiplier and also handles any fused branches (which must be placed into the last slot in a dispatch group).

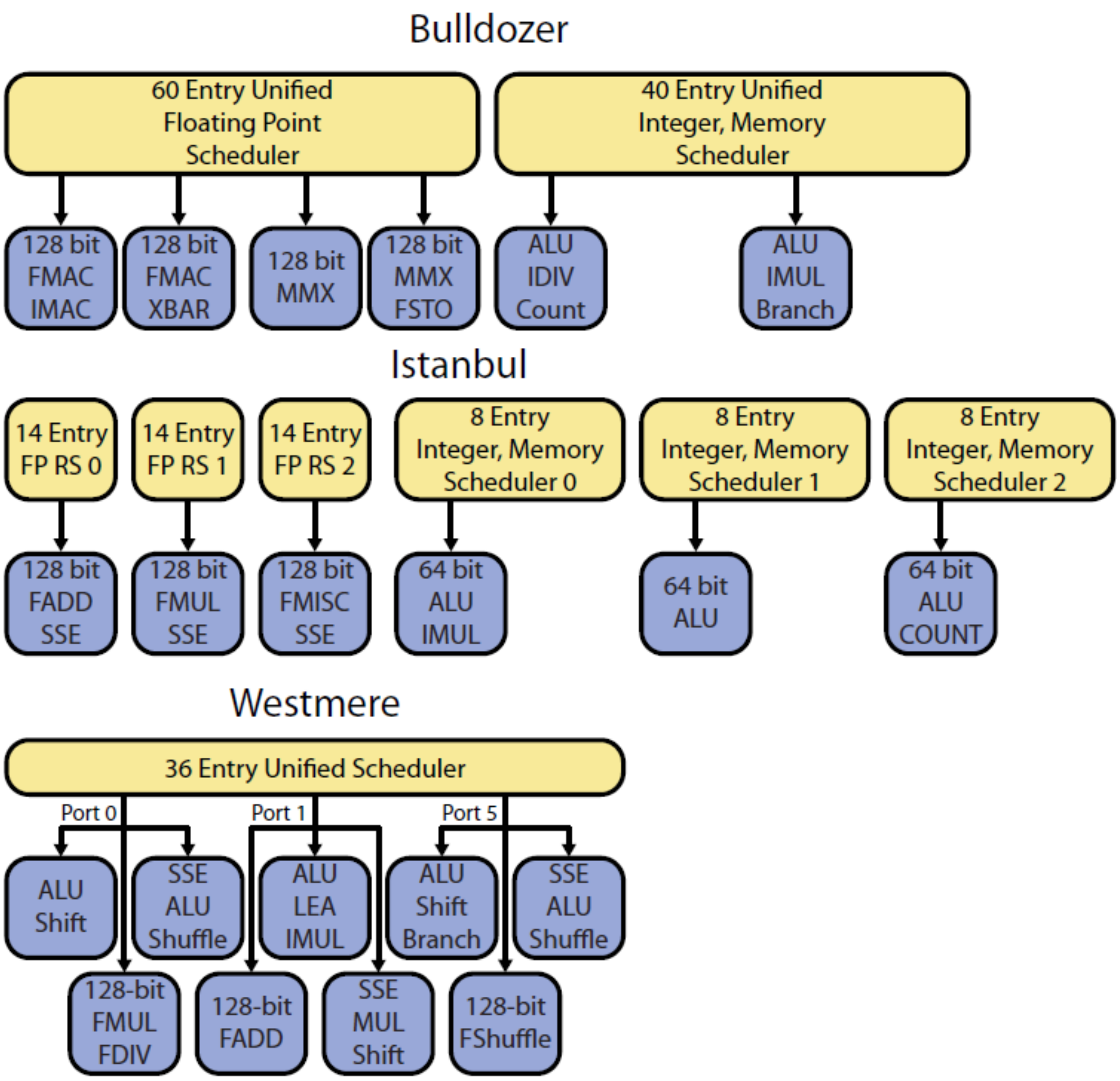


Figure 5 – Bulldozer Execution Units and Comparison

Shared Floating Point Execution Units

While Bulldozer can execute the new AVX instructions, all of the execution units are 128 bits wide. Thus it is extremely likely that any 256-bit instructions must be executed as two uops and are decoded as two macro-ops (since a macro-op cannot contain two execution uops). This would be consistent with AMD's conservative approach to embracing instruction set extensions from Intel. For example, while Intel introduced SSE2 with the P4 in 2000, it took until the K8 in 2003 for AMD to support the new instructions. Similarly, Intel went to 128-bit execution units in 2006 and it wasn't until Barcelona in 2007 that AMD caught up. The rationale for AMD's slower uptake is straight forward. New instructions are not immediately put to use by most software vendors – so even though Sandy Bridge will arrive in late 2010 or early 2011, most software will not use AVX. Consequently, it does

not make sense for AMD to dedicate the die area, design effort and power until the software really does catch up.
Like the integer cores, Bulldozer’s floating point cluster does away with the notion of dedicated schedulers and lanes and uses the more flexible unified approach. The four pipelines (P0-P3), are fed from a shared 60 entry scheduler. This is roughly 50% larger than the reservation stations for Istanbul (42 entries) and almost double Barcelona’s (36 entries). The heart of the cluster is a pair of 128-bit wide floating point multiply-accumulate (FMAC) units on P0 and P1. Each FMAC unit also handles division and square root operations with variable latency.
The two FMAC units execute FADD and FMUL instructions, although this obviously leaves performance on the table compared to using the combined FMAC. The first pipeline includes a 128-bit integer multiply-accumulate unit, primarily used for instructions in AMD’s XOP. Additionally, the hardware for converting between integer and floating point data types is tied to pipeline 0. Pipeline 1 also serves double duty and contains the crossbar hardware (XBAR) used for permutations, shifts, shuffles, packing and unpacking.
Another question regarding Bulldozer is how 256-bit AVX instructions are handled by the execution units. One option is to treat each half as a totally independent macro-op, as the K8 did for 128-bit SSE, and let the schedulers sort everything out. However, it is possible that Bulldozer’s two symmetric FMAC units could be ganged together to execute both halves of an AVX instruction simultaneously to reduce latency.
The other half of the floating point cluster’s execution units actually have little to do with floating point data at all. Bulldozer has a pair of largely symmetric 128-bit integer SIMD ALUs (P2 and P3) that execute arithmetic and logical operations. P3 also includes the store unit (STO) for the floating point cluster (this was called the FMISC unit in Istanbul). Despite the name, it does not actually perform stores – rather it passes the data for the store to the load-store unit, thus acting as a conduit to the actual store pipeline.
In a similar fashion, there is a small floating point load buffer (not shown above) which acts as an analogous conduit for loads between the load-store units and the FP cluster. The FP cluster can execute two 128-bit loads per cycle, and one of the purposes of the FP load buffer is to smooth the bandwidth between the two cores. For example, if the two cores simultaneously send data for four 128-bit loads to the FP cluster, the buffer would release 256-bits of data in the first cycle, and then 256-bits of data in the next cycle.

Memory Subsystem

Perhaps the most profound change in Bulldozer is the load-store pipeline and caches. Other sections of the chip were rearchitected for efficiency or for modest performance gains. However, the load store units were totally redesigned and improved across the board. In tandem, the inner portions of the cache hierarchy have been redone, and for the first time, AMD is fielding competitive prefetchers.

The memory pipeline for each Bulldozer core starts with the load and store queues and the integer scheduler. Any loads or stores in flight must be allocated an entry in the appropriate memory queues. This is necessary to maintain the relatively strong x86 memory ordering model. Previously, Istanbul had a somewhat complex two level load-store queue, where different functions were performed in each level. Bulldozer has a conceptually simpler microarchitecture with a separate 40 entry load queue, and a 24 entry store queue. In total, this means that each Bulldozer core can have 33% more memory operations in flight compared to the previous generation and about 20-30% less than Nehalem or Westmere.

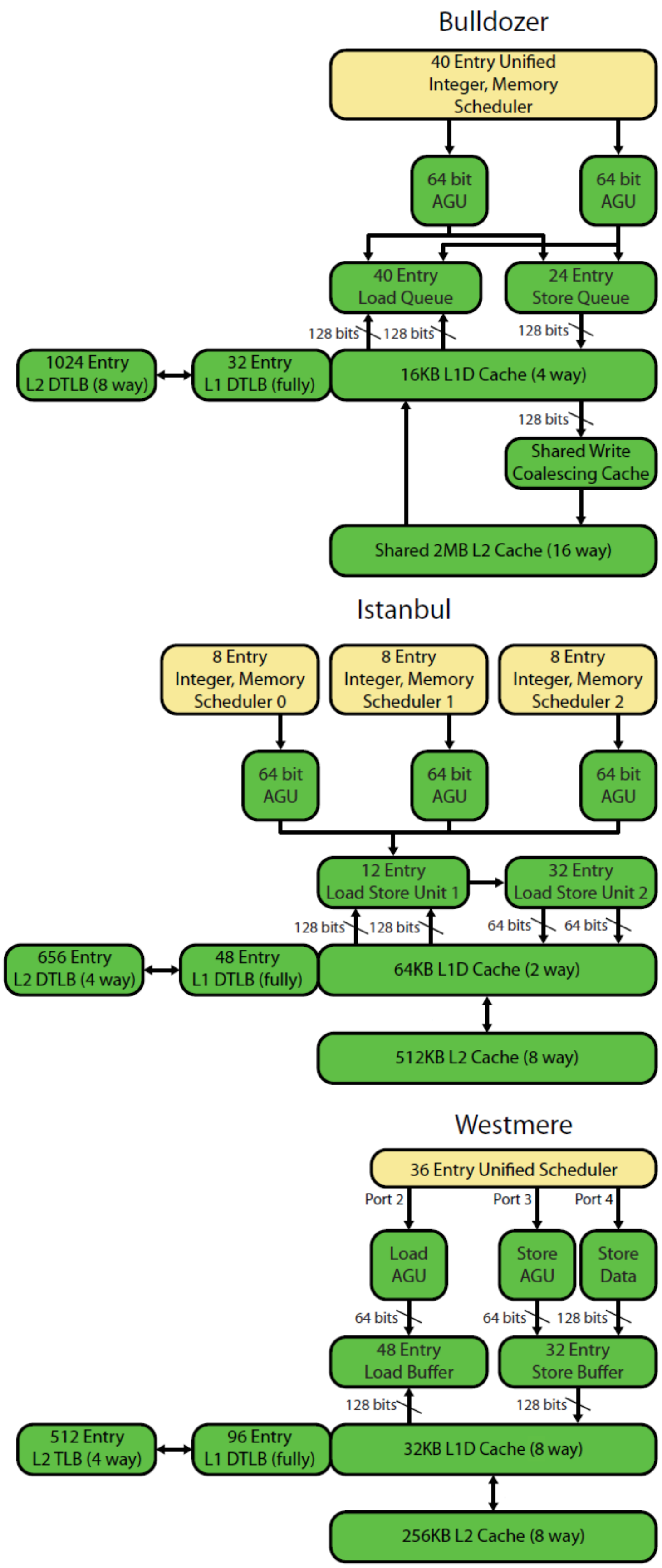

Bulldozer
40 Entry Unified Integer, Memory Scheduler
64 bit AGU
64 bit AGU
40 Entry Load Queue
24 Entry Store Queue
128 bits
128 bits
128 bits
1024 Entry L2 DTLB (8 way)
32 Entry L1 DTLB (fully)
16KB L1D Cache (4 way)
128 bits
Shared Write Coalescing Cache
Shared 2MB L2 Cache (16 way)
Istanbul
8 Entry Integer, Memory Scheduler 0
8 Entry Integer, Memory Scheduler 1
8 Entry Integer, Memory Scheduler 2
64 bit AGU
64 bit AGU
64 bit AGU
12 Entry Load Store Unit 1
32 Entry Load Store Unit 2
128 bits
128 bits
64 bits
64 bits
656 Entry L2 DTLB (4 way)
48 Entry L1 DTLB (fully)
64KB L1D Cache (2 way)
512KB L2 Cache (8 way)
Westmere
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 L1D Cache (8 way)
256KB L2 Cache (8 way)

Figure 6 – Bulldozer Memory Subsystem and Comparison

The scheduler feeds memory operations into the two AGUs responsible for address generation. While this is a decrease from the prior generation, there are reasons to suspect this may not be catastrophic. The original K8 had a totally in-order memory pipeline, while Istanbul had a non-speculative out-of-order memory pipeline – loads could only move ahead of stores known to have a different address. Bulldozer improves this further with a dependence predictor that will determine when loads can speculatively pass stores. This latter technique is referred to as memory disambiguation by Intel and first showed up in the Core 2 Duo. Second, some macro-ops are of the form 'load-op-store' and only do address generation and translation for the load, and re-use that work for the ending store. Third, it's possible that for 256-bit AVX instructions, the address generation is only done once, and that the second macro-op simply adds a 16B offset to the address of the first macro-op.

Once a memory access is ready to proceed, it probes the 32-entry, fully associative L1 DTLB for address translation. Misses in the DTLB will access the 1024-entry, 8-way associative L2 DTLB, and both TLBs can cache any combination of 4KB, 2MB and 1GB pages. For the L2 DTLB, this is a big improvement in both size and flexibility over Istanbul, where the L2 DTLB could hold 512 4KB pages, 128 2MB pages and 16 1GB pages.

The cache hierarchy has also taken a radical change with Bulldozer and to fully understand the architecture it is necessary to look at both the L1D and L2 caches. Bulldozer's 16KB L1D cache is a 4-way associative, write-through design with 64B lines and a 4 cycle load-to-use latency. It is virtually indexed and physically tagged, so that the TLB look up can proceed in parallel with checking the index. Normally in a 4-way cache, an access must check 4 different locations (one per way) simultaneously to find the requested data. Bulldozer's L1D uses way prediction to save power by predicting which of the 4 ways will contain the data and checking that way first. The downside is that a misprediction costs one or more cycles of added latency (AMD did not disclose this penalty). Bulldozer's load-to-use latency increased by one cycle over Istanbul, and is now identical to Nehalem's. This change in timing reflects AMD's high frequency targets and high bandwidth requirements for the L1D cache. The cache is banked for simultaneous accesses, although the arrangement was not disclosed. Given that three accesses are possible each cycle, 8 banks seems too small, 32 banks too large, and 16 banks – that sounds just right. The L1D is theoretically capable of two 128-bit loads and one 128-bit store per cycle (48B/cycle), although bank conflicts may reduce the available bandwidth. Moreover, the L1D cache cannot sustain three independent accesses per cycle since there are only two AGUs. However, the extra port is beneficial for clearing out queued up operations and probably also plays a critical role for executing 256-bit AVX instructions. In the context of AVX, Bulldozer probably has equivalent cache throughput to Sandy Bridge – undoubtedly one of the major design targets. The L1D bandwidth is a substantial step forward for Bulldozer, as Istanbul was limited to either 2×128-bit loads or 2×64-bit stores or one of each.

Bulldozer's L2 cache is shared between the two cores in a module and is mostly inclusive of the L1D caches (recall that the L1D is write-through). The size is implementation dependent and early versions will be either 1MB or 2MB. Open64 compiler optimization notes indicate that Interlagos probably uses a 2MB and 16-way associative design. The load-to-use latency for Bulldozer is surprisingly high: 18-20 cycles, again reflecting a focus on high frequency. In comparison, the L2 caches for Nehalem and Istanbul are roughly 10 cycle latency, although the capacities are smaller (256KB and 512KB respectively). The L2 cache can have as many as 23 outstanding misses concurrently, which is a somewhat peculiar number compared to the usual powers of 2. This suggests that some outstanding miss requests may be dedicated for certain purposes. For example, there might be 8 misses outstanding for each L1D cache, with the remainder for use by the L1I cache and prefetchers.

Memory Subsystem Continued

Since the L1D is both write-through and mostly included in the L2, evicting a cache line from the L1D is silent and requires no further actions. This is beneficial since evictions are typically caused by a filling a cache line, in response to a cache miss and closely tied to the critical path for a miss. In the exclusive L1D cache for Istanbul, moving the evicted line from L1D to L2 contributed to the latency for a cache miss.

The relationship between the L1D and L2 caches also simplifies reliability. Since any data written by the L1D is also present in the L2, parity is sufficient protection for the L1; any errors can be fixed by reloading from the ECC protected L2 (or L3/memory). As a result, ECC is no longer required for the L1D (as it was for Istanbul), which reduces the power consumption for stores. In Istanbul, any store to a cache line had to first read to get the ECC, then recalculate the ECC with the new data and then finally write to the cache.

While the L1D is mostly included in the L2, there are some situations where lines can reside in the L1D without being present in the L2. As a result, the L1D may need to be snooped when another core misses in the L3 cache. This is extremely undesirable, since there will be substantial snoop traffic in a Bulldozer system which will cost both power and performance if the L1D caches must always be snooped by remote cache misses. In Nehalem, the L3 cache is inclusive precisely to eliminate this sort of snoop traffic. It stands to reason that Bulldozer was designed to eliminate snoop traffic to the L1D caches and instead have the L2 cache for each module handle all the coherency snoops for that module. Unfortunately, AMD was unwilling to disclose the precise nature of their coherency protocol at Hot Chips, so we will have to wait to find out more details.

One disadvantage of a write-through policy is that the L1D caches do not insulate the L2 cache from the store traffic in the cache hierarchy. Consequently, the L2 cache must have higher bandwidth to accommodate all the store traffic from two cores, and any associated snoop traffic and responses.

To alleviate the write-through bandwidth requirements on the L2, each Bulldozer module includes a write coalescing cache (WCC), which is considered part of the L2. At present, AMD has not disclosed the size and associativity of the WCC, although it is probably quite small. Stores from both

L1D caches go through the WCC, where they are buffered and coalesced. The purpose of the WCC is to reduce the number of writes to the L2 cache, by taking advantage of both spatial and temporal locality between stores. For example, a memcpy() routine might clear a cache line with four 128-bit stores, the WCC would coalesce these stores together and only write out once to the L2 cache.

Most implementations of Bulldozer (certainly Interlagos) will share a single L3 cache, which acts as a mostly exclusive victim buffer for the L2 caches in each module. Again, AMD would not disclose any information as it concerns the overall product, rather than the core itself. However, it is possible to make an intelligent estimate based on public information. Assuming that each Interlagos die will have 8MB of L2 cache for 4 modules, the L3 is most likely to be 8MB.

AMD cannot afford to produce a die with 16MB of L3 cache on 32nm and 4MB is probably too small. When Barcelona was first released on 65nm, the L3 cache was 2MB – equal to the aggregate size of the four L2 caches. It seems reasonable that AMD would return to this arrangement. The associativity is an open question, but should be at least 16-way and more likely 32 or 64 way. It is also expected that AMD has further refined and improved the sharing and contention management policies in the L3 cache.

Prefetching is another area where historically Intel has relentlessly focused, and AMD has lagged behind. Prefetching can be highly effective at reducing memory latency, and can lead to tremendous increases in performance – especially for workloads with complex data structures that tend to incur many cache misses. In Bulldozer, there was a tremendous amount of effort put into the prefetching that should yield good results. The exact nature of the strided prefetchers (i.e. where addresses are offset by exactly +/-N bytes) was not discussed, but that is an area which has been very thoroughly explored in academia and commercial products.

More intriguing is Bulldozer's non-strided data prefetcher, which is useful for accessing more complex and irregular data structures (e.g. linked lists, B-trees, etc.). Again, AMD did not disclose their approach, but one possibility is what we might describe as a 'pattern history based prefetcher'. The prefetcher tracks the addresses of misses and tries to identify specific patterns of misses that occur together (temporally). Once a pattern of misses has been detected, the prefetcher will find the first miss in the pattern. When this first miss occurs again, the prefetcher will immediately prefetch the rest of the pattern. For traversing a complex data structure like a linked list, this would be a fairly effective approach. There are other techniques that have been discussed in academic literature, and it will be interesting to see which AMD implemented.

Power Management

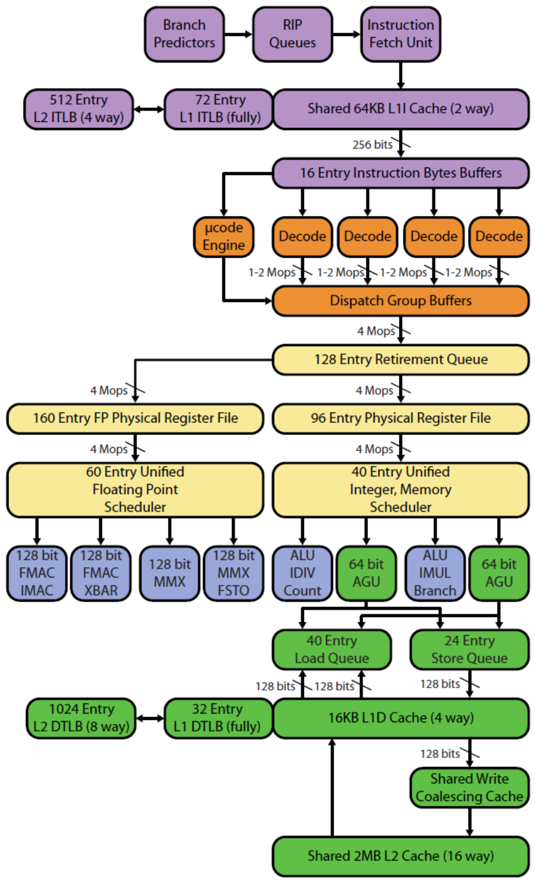
At ISSCC 2010, AMD disclosed considerable information about Llano – AMD's other CPU manufactured on Global Foundries' 32nm SOI process. Llano will have a dynamic voltage and frequency scaling (DVFS) system and core level power gating. Thus it should come as no surprise that these features also made their way into Bulldozer implementations. The actual techniques for

Bulldozer will probably be described at next year's ISSCC, but for now, we can intelligently speculate that any technique used in Llano may show up in Bulldozer, possibly with improvements.

Llano's DVFS is described as relying upon a digital activity measurement approach. This family of techniques was pioneered by Intel's Fort Collins circuit design team for Tukwila, several members of which subsequently left Intel to join AMD. The approach in Llano samples 95 performance counter signals that are closely correlated with switching capacitance, using scan chain-like hardware. These samples are used to estimate dynamic power within 2% accuracy. The power estimates in turn are used to dynamically increase frequency based on available power headroom. It is expected that Bulldozer will use similar techniques, although perhaps enhanced to take advantage of certain microarchitectural differences and further refined with additional design effort.

Llano is also the first announced CPU from AMD to use power gating. The power gates are implemented as a footer ring of NFETs around the periphery of a core and L2 cache, using the package plane as a virtual ground. While Bulldozer's hierarchical and shared microarchitecture improves area efficiency and throughput, it does complicate power gating. Conceptually there are five major circuit regions of each Bulldozer module – the shared front-end, the two integer cores, the floating point cluster and the L2 cache. Unfortunately, if a single core is active all of these regions (perhaps save the other integer core) must be active. The benefit of power gating a lone integer core is not worth the complexity of the implementation problems, especially since the operating system scheduler should be power-aware. As a result, Bulldozer's granularity of power gating is at the module level. Each Interlagos die incorporates at least 4 power gates, one for each Bulldozer module. In a server, all the memory controllers must stay active to service any cache misses from other chips. Thus it is an open question whether there are substantial benefits to power gating the entire northbridge – i.e. L3 cache and other shared components such as interconnects and memory controllers. The server oriented Nehalem only power gated the cores, while Westmere (intended for desktops and notebooks) had a separate power gate for the uncore/northbridge.

Bulldozer



Westmere

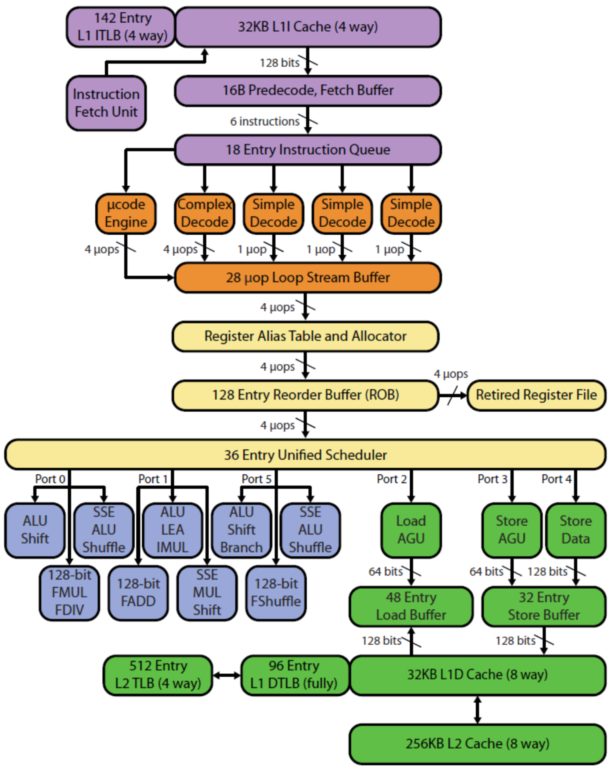


Figure 7 – Bulldozer and Westmere Microarchitectures

Conclusion

For AMD, Interlagos – the first Bulldozer implementation – is their next chance to regain parity or retake the lead in the server market with Intel. The philosophy behind Bulldozer is to step back from trying to go toe-to-toe with Intel, which is good, since that approach has never worked for AMD. Instead, Bulldozer focuses on a more server-centric set of goals and tries to re-evaluate various trade-offs in that context. Figure 7 above is a comparison of Bulldozer's microarchitecture to the more conventional Westmere that shows some of the differences between the two.

The Bulldozer architecture is fairly innovative, especially by the standards of x86 CPU designs. The hierarchical sharing and high frequency design should help AMD achieve higher performance per mm^2 of silicon – this is necessary for success, since Intel is roughly 12-18 months ahead of AMD's manufacturing partner. The ultimate question is how does the novel architecture in Bulldozer translate into performance (both single threaded and multi-threaded), power and die area. Most of the physical characteristics of Interlagos, such as frequency and die area, are unknown. This leaves a great deal of uncertainty, as performance is highly dependent upon frequency as well as many of the details of the architecture which have been withheld for competitive reasons. The biggest questions about Bulldozer are the frequency, branch prediction, various queues and buffers in the front-end, handling of 256-bit AVX instructions, coherency protocol and northbrige microarchitecture. Over the next year, AMD will incrementally release more information – at their analyst day, ISSCC and other venues. These details should give a fuller picture of Bulldozer and what to expect from productizations such as Interlagos. In the meantime, the team behind Bulldozer deserves congratulations for nearing the finish line on the most novel and complicated design in AMD's history. We look forward to seeing the results next year.

References

[1] Butler, Mike. "Bulldozer" A new approach to multithreaded compute performance. Hot Chips XXII, August 2010.

[2] GCC Mailing list discussion. http://gcc.gnu.org/ml/gcc/2010-06/msg00402.html

[3] Open64 config_cache_targ.cxx http://svn.open64.net/filedetails.php?repname=Open64&path=/trunk/osprey/common/com/x8664/config_cache_targ.cxx

[4] Comp.arch discussion. http://groups.google.com/group/comp.arch/browse_frm/thread/45018bf3214f6049?hl=en#

[5] Interview with Mike Butler, Chuck Moore, Gary Silcott.

[6] Jotwani, R. et al. "An x86-64 Core Implemented in 32nm SOI CMOS," Proceedings of International Solid State Circuits Conference, pp 106-107, February 2010.

[7] Conway, P. et al. Blade Computing with the AMD Opteron Processor ("Magny-Cours"). Hot Chips XXI, August 2009.