White Paper

Inside Intel® Core™ Microarchitecture and Smart Memory Access

An In-Depth Look at Intel Innovations for Accelerating Execution of Memory-Related Instructions

Jack Doweck
Intel Principal Engineer,
Merom Lead Architect
Intel Corporation

Introduction

The Intel® Core™ microarchitecture is a new foundation for Intel® architecture-based desktop, mobile, and mainstream server multi-core processors. This state-of-the-art, power-efficient multi-core microarchitecture delivers increased performance and performance per watt, thus increasing overall energy efficiency. Intel Core microarchitecture extends the energy-efficient philosophy first delivered in Intel's mobile microarchitecture (Intel® Pentium® M processor), and greatly enhances it with many leading edge microarchitectural advancements, as well as some improvements on the best of Intel NetBurst® microarchitecture. This new microarchitecture also enables a wide range of frequencies and thermal envelopes to satisfy different needs.

With its higher performance and low power, the new Intel Core microarchitecture and the new processors based on it will inspire many new computers and form factors. These processors include, for desktops, the new Intel® Core™2 Duo processor. This advanced desktop processor is expected to power higher performing, ultra-quiet, sleek, and low-power computer designs and new advances in sophisticated, user-friendly entertainment systems. For mainstream enterprise servers, the new Intel® Xeon® server processors are expected to reduce space and electricity burdens in server data centers, as well as increase responsiveness, productivity, and energy efficiency across server platforms. For mobile users, the new Intel® Centrino® Duo mobile technology featuring the Intel Core 2 Duo processor will mean greater computer performance and new achievements in enabling leading battery life in a variety of small form factors for world-class computing "on the go."

This paper provides a brief look at the five major "ingredients" of the Intel Core microarchitecture. It then dives into a deeper explanation of the paper's main topic, the key innovations comprising Intel® Smart Memory Access.

The Five Major Ingredients of Intel Core Microarchitecture

Five main ingredients provide key contributions to the major leaps in performance and performance-per-watt delivered by the Intel Core microarchitecture.

These ingredients are:

- Intel® Wide Dynamic Execution
- Intel® Advanced Digital Media Boost
- Intel® Intelligent Power Capability
- Intel® Advanced Smart Cache
- Intel Smart Memory Access

Together, these features add up to a huge advance in energy-efficient performance. The Intel Core 2 Duo desktop processor, for example, delivers more than 40 percent improvement in performance and a greater than 40 percent reduction in power as compared to today's high-end Intel® Pentium® D processor 950. (Performance based on estimated SPECint*_rate_base2000. Actual performance may vary. Power reduction based on TDP.)[1] Intel mobile and server processors based on this new micro-architecture provide equally impressive gains.

Intel® Wide Dynamic Execution

Dynamic execution is a combination of techniques (data flow analysis, speculative execution, out of order execution, and super scalar) that Intel first implemented in the P6 microarchitecture used in the Intel® Pentium® Pro processor, Pentium® II processor and Pentium® III processors.

Intel Wide Dynamic Execution significantly enhances dynamic execution, enabling delivery of more instructions per clock cycle to improve execution time and energy efficiency. Every execution core is 33 percent wider than previous generations, allowing each core to fetch, decode, and retire up to four full instructions simultaneously. However, to maximize performance on common mixes of instructions received from programs, the execution core can dispatch and execute at a rate of five instructions per cycle for either mixes of three integer instructions, one load and one store; or mixes of two floating point/vector instructions, one integer instruction, one load and one store.

Intel Wide Dynamic Execution also includes a new and innovative capability called Macrofusion. Macrofusion combines certain common x86 instructions into a single instruction that is executed as a single entity, increasing the peak throughput of the engine to five instructions per clock. The wide execution engine, when Macro-fusion comes into play, is then capable of up to six instructions per cycle throughputs for even greater energy-efficient performance. Intel Core microarchitecture also uses extended microfusion, a technique that "fuses" micro-ops derived from the same macro-op to reduce the number of micro-ops that need to be executed. Studies have shown that micro-op fusion can reduce the number of micro-ops handled by the out-of-order logic by more than 10 percent. Intel Core microarchitecture "extends" the number of micro-ops that can be fused internally within the processor.

Intel Core microarchitecture also incorporates an updated ESP (Extended Stack Pointer) Tracker. Stack tracking allows safe early resolution of stack references by keeping track of the value of the ESP register. About 25 percent of all loads are stack loads and 95 percent of these loads may be resolved in the front end, again contributing to greater energy efficiency [Bekerman].

Micro-op reduction resulting from micro-op fusion, Macrofusion, ESP Tracker, and other techniques make various resources in the engine appear virtually deeper than their actual size and results in executing a given amount of work with less toggling of signals—two factors that provide more performance for the same or less power.

Intel Core microarchitecture also provides deep out-of-order buffers to allow for more instructions in flight, enabling more out-of-order execution to better exploit instruction-level parallelism.

1. Please refer to *www.intel.com/performance* for all performance related claims.

Intel® Advanced Digital Media Boost

Intel Advanced Digital Media Boost helps achieve similar dramatic gains in throughputs for programs utilizing SSE instructions of 128-bit operands. (SSE instructions enhance Intel architecture by enabling programmers to develop algorithms that can mix packed, single-precision, and double-precision floating point and integers, using SSE instructions.) These throughput gains come from combining a 128-bit-wide internal data path with Intel Wide Dynamic Execution and matching widths and throughputs in the relevant caches. Intel Advanced Digital Media Boost enables most 128-bit instructions to be dispatched at a throughput rate of one per clock cycle, effectively doubling the speed of execution and resulting in peak floating point performance of 24 GFlops (on each core, single precision, at 3 GHz frequency). Intel Advanced Digital Media Boost is particularly useful when running many important multimedia operations involving graphics, video, and audio, and processing other rich data sets that use SSE, SSE2, and SSE3 instructions.

Intel® Intelligent Power Capability

Intel Intelligent Power Capability is a set of capabilities for reducing power consumption and device design requirements. This feature manages the runtime power consumption of all the processor's execution cores. It includes an advanced power-gating capability that allows for an ultra fine-grained logic control that turns on individual processor logic subsystems only if and when they are needed. Additionally, many buses and arrays are split so that data required in some modes of operation can be put in a low-power state when not needed. In the past, implementing such power gating has been challenging because of the power consumed in powering down and ramping back up, as well as the need to maintain system responsiveness when returning to full power [Wechsler]. Through Intel Intelligent Power Capability, Intel has been able to satisfy these concerns, ensuring significant power savings without sacrificing responsiveness.

Intel® Advanced Smart Cache

Intel Advanced Smart Cache is a multi-core optimized cache that improves performance and efficiency by increasing the probability that each execution core of a dual-core processor can access data from a higher-performance, more-efficient cache subsystem. To accomplish this, Intel Core microarchitecture shares the Level 2 (L2) cache between the cores. This better optimizes cache resources by storing data in one place that each core can access. By sharing L2 cache between each core, Intel Advanced Smart Cache allows each core to dynamically use up to 100 percent of available L2 cache. Threads can then dynamically use the required cache capacity. As an extreme example, if one of the cores is inactive, the other core will have access to the full cache. Intel Advanced Smart Cache enables very efficient sharing of data between threads running in different cores. It also enables obtaining data from cache at higher throughput rates for better performance. Intel Advanced Smart Cache provides a peak transfer rate of 96 GB/sec (at 3 GHz frequency).

Intel® Smart Memory Access

Intel Smart Memory Access improves system performance by optimizing the use of the available data bandwidth from the memory subsystem and hiding the latency of memory accesses. The goal is to ensure that data can be used as quickly as possible and is located as close as possible to where it's needed to minimize latency and thus improve efficiency and speed. Intel Smart Memory Access includes a new capability called memory disambiguation, which increases the efficiency of out-of-order processing by providing the execution cores with the built-in intelligence to speculatively load data for instructions that are about to execute before all previous store instructions are executed. Intel Smart Memory Access also includes an instruction pointer-based prefetcher that "prefetches" memory contents before they are requested so they can be placed in cache and readily accessed when needed. Increasing the number of loads that occur from cache versus main memory reduces memory latency and improves performance.

How Intel Smart Memory Access Improves Execution Throughput

The Intel Core microarchitecture Memory Cluster (also known as the Level 1 Data Memory Subsystem) is highly out-of-order, non-blocking, and speculative. It has a variety of methods of caching and buffering to help achieve its performance. Included among these are Intel Smart Memory Access and its two key features: memory disambiguation and instruction pointer-based (IP-based) prefetcher to the level 1 data cache.

To appreciate how memory disambiguation and instruction pointer-based prefetcher to the level 1 data cache improve execution throughput, it's important to understand that typical x86 software code contains about 38 percent memory stores and loads. Generally there are twice as many loads as there are stores. To prevent data inconsistency, dependent memory-related instructions are normally executed in the same order they appear on the program. This means if a program has an instruction specifying a "store" at a particular address and then a "load" from that same address, these instructions have to be executed in that order. But what about all the stores and loads that don't share the same address? How can their non-dependence on each other be used to improve processing efficiency and speed?

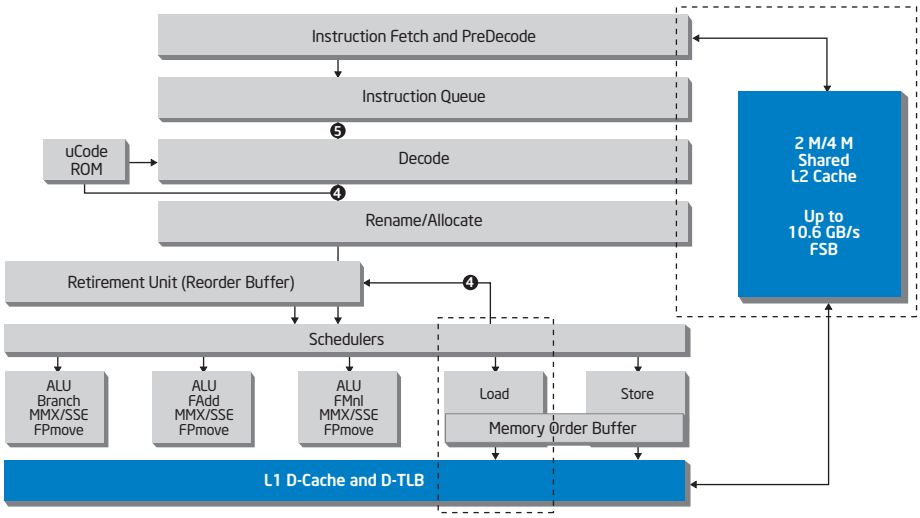


Figure 1: Intel Smart Memory Access uses memory disambiguation and a number of prefetchers (including an instruction pointer-based prefetcher to the level 1 data cache) to help Intel Core microarchitecture achieve its high levels of performance.

Memory Disambiguation

Since the Intel Pentium Pro, all Intel processors have featured a sophisticated out-of-order memory engine allowing the CPU to execute non-dependent instructions in any order. But they had a significant shortcoming. These processors were built around a conservative set of assumptions concerning which memory accesses could proceed out of order. They would not move a load in the execution order above a store having an unknown address (cases where a prior store has not been executed yet). This was because if the store and load end up sharing the same address, it results in an incorrect instruction execution. Yet many loads are to locations unrelated to recently executed stores. Prior hardware implementations created false dependencies if they blocked such loads based on unknown store addresses. All these false dependencies resulted in many lost opportunities for out-of-order execution.

In designing Intel Core microarchitecture, Intel sought a way to eliminate false dependencies using a technique known as memory disambiguation. ("Disambiguation" is defined as the clarification that follows the removal of an ambiguity.) Through memory disambiguation, Intel Core microarchitecture is able to resolve many of the cases where the ambiguity of whether a particular load and store share the same address thwart out-of-order execution.

Memory disambiguation uses a predictor and accompanying algorithms to eliminate these false dependencies that block a load from being moved up and completed as soon as possible. The basic objective is to be able to ignore unknown store-address blocking conditions whenever a load operation dispatched from the processor's reservation station (RS) is predicted to not collide with a store. This prediction is eventually verified by checking all RS-dispatched store addresses for an address match against newer loads that were predicted non-conflicting and already executed. If there is an offending load already executed, the pipe is flushed and execution restarted from that load.

The memory disambiguation predictor is based on a hash table that is indexed with a hashed version of the load's EIP address bits. ("EIP" is used here to represent the instruction pointer in all x86 modes.) Each predictor entry behaves as a saturating counter, with reset.

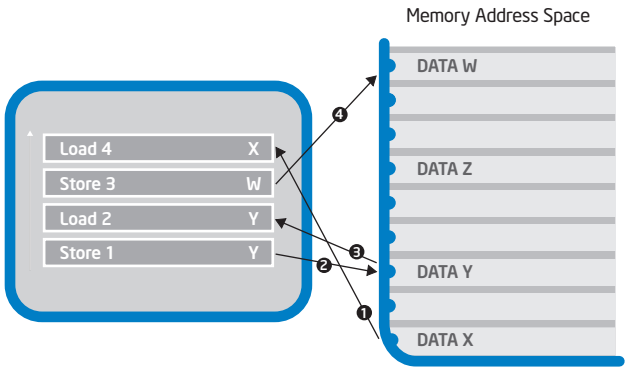


Figure 2: In this example of memory disambiguation, the circled numbers on the arrows indicate chronological execution order and the arrow on the far left shows program order. As you can see, Load 2 cannot be moved forward since it has to wait until Store 1 is executed so variable Y has its correct value. However, Intel's memory disambiguation predictor can recognize that Load 4 isn't dependent on the other instructions shown and can be executed first without having to wait for either Store 3 or Store 1 to execute. By executing Load 4 several cycles earlier, the CPU now has the data required for executing any instructions that need the value of X, thus reducing memory latency and delivering a higher degree of instruction-level parallelism.

The predictor has two write operations, both done during the load's retirement:

1. Increment the entry (with saturation at maximum value) if the load "behaved well." That is, if it met unknown store addresses, but none of them collided.
2. Reset the entry to zero if the load "misbehaved." That is, if it collided with at least one older store that was dispatched by the RS after the load. The reset is done regardless of whether the load was actually disambiguated.

The predictor takes a conservative approach. In order to allow memory disambiguation, it requires that a number of consecutive iterations of a load having the same EIP behave well. This isn't necessarily a guarantee of success though. If two loads with different EIPs clash in the same predictor entry, their prediction will interact.

Predictor Lookup

The predictor is looked up when a load instruction is dispatched from the RS to the memory pipe. If the respective counter is saturated, the load is assumed to be safe. The result is written to a "disambiguation allowed bit" in the load buffer. This means that if the load finds a relevant unknown store address, this condition is ignored and the load is allowed to go on. If the predictor is not saturated, the load will behave like in prior implementations. In other words, if there is a relevant unknown store address, the load will get blocked.

Load Dispatch

In case the load meets an older unknown store address, it sets the "update bit" indicating the load should update the predictor. If the prediction was "go," the load will be dispatched and set the "done" bit indicating that disambiguation was done. If the prediction was "no go," the load will be conservatively blocked until resolving of all older store addresses.

Prediction Verification

To recover in case of a misprediction by the disambiguation predictor, the address of all the store operations dispatched from the RS to the Memory Order Buffer must be compared with the address of all the loads that are younger than the store. If such a match is found the respective "reset bit" is set. When a load retires that was disambiguated and its reset bit set, we restart the pipe from that load to re-execute it and all its dependent instructions correctly.

Watchdog Mechanism

Obviously, since disambiguation is based on prediction and mispredictions can cause execution pipe flush, it's important to build in safeguards to avoid rare cases of performance loss. Consequently, Intel Core microarchitecture includes a mechanism to temporarily disable memory disambiguation to prevent cases of performance loss. This mechanism constantly monitors the success rate of the disambiguation predictor.

Instruction Pointer-Based (IP) Prefetcher to Level 1 Data Cache

In addition to memory disambiguation, Intel Smart Memory Access includes advanced prefetchers. Just like their name suggests, prefetchers "prefetch" memory data before it's requested, placing this data in cache for "just-in-time" execution. By increasing the number of loads that occur from cache versus main memory, prefetching reduces memory latency and improves performance.

The Intel Core microarchitecture includes in each processing core two prefetchers to the Level 1 data cache and the traditional prefetcher to the Level 1 instruction cache. In addition it includes two prefetchers associated with the Level 2 cache and shared between the cores. In total, there are eight prefetchers per dual core processor.

Of particular interest is the IP-based prefetcher that prefetches data to the Level 1 data cache. While the basic idea of IP-based prefetching isn't new, Intel made some microarchitectural innovations to it for Intel Core microarchitecture.

The purpose of the IP prefetcher, as with any prefetcher, is to predict what memory addresses are going to be used by the program and deliver that data just in time. In order to improve the accuracy of the prediction, the IP prefetcher tags the history of each load using the Instruction Pointer (IP) of the load. For each load with an IP, the IP prefetcher builds a history and keeps it in the IP history array. Based on load history, the IP prefetcher tries to predict the address of the next load accordingly to a constant stride calculation (a fixed distance or "stride" between subsequent accesses to the same memory area). The IP prefetcher then generates a prefetch request with the predicted address and brings the resulting data to the Level 1 data cache.

Obviously, the structure of the IP history array is very important here for its ability to retain history information for each load. The history array in the Intel Core microarchitecture consists of following fields:

- 12 untranslated bits of last demand address
- 13 bits of last stride data (12 bits of positive or negative stride with the 13th bit the sign)
- 2 bits of history state machine
- 6 bits of last prefetched address—used to avoid redundant prefetch requests

Using this IP history array, it's possible to detect iterating loads that exhibit a perfect stride access pattern ($A_n - A_{n-1}$ = Constant) and thus predict the address required for the next iteration. A prefetch request is then issued to the L1 cache. If the prefetch request hits the cache, the prefetch request is dropped. If it misses, the prefetch request propagates to the L2 cache or memory.

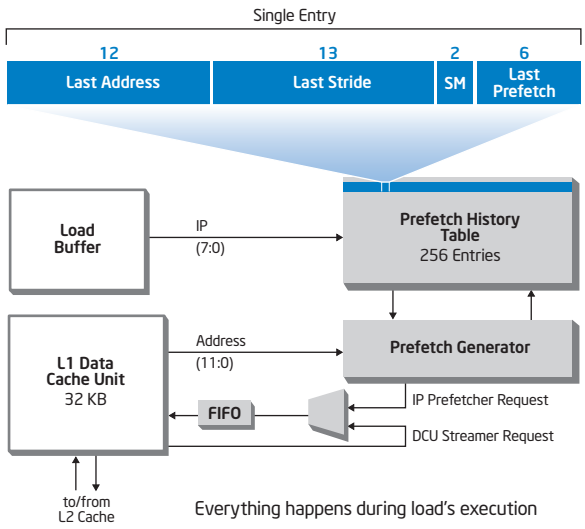


Figure 3: High level block diagram of the relevant parts in the Intel Core microarchitecture IP prefetcher system.

Traffic Control and Resource Allocation

Among the important considerations a prefetcher design needs to answer is how to minimize possible side effects (such as overloading of resources) of prefetching. The Intel Core microarchitecture IP prefetcher includes a number of measures to mitigate these side effects.

The prefetch request generated by the prefetcher goes to a First In/First Out (FIFO) buffer where it waits for "favorable conditions" to issue a prefetch request to the L1 cache unit. A prefetch request then moves to the store port of the cache unit and is removed from the FIFO when:

1. The store port is idle.
2. There are at least a set number of Fill Buffer entries empty.
3. There are at least a set number of entries empty in the external bus queue.
4. The cache unit was able to accept the request.

Upon successful reception of a prefetch request by the data cache unit, a lookup for that line is performed in the cache and Fill Buffers. If the prefetch request hits, it is dropped. Otherwise a corresponding line read request is generated to the Level 2 cache or bus just as a normal demand load miss would have done. In particular, the L2 prefetcher treats prefetch requests just as a demand request.

What happens if the prefetch FIFO is full? New requests override the oldest entries.

One interesting advantage of the Intel Core microarchitecture is that the parameters for prefetch traffic control and allocation can be fine-tuned for the platform. Products can have their prefetching sensitivity set based on chipset memory, FSB speed, L2 cache size, and more. In addition, since server products run radically different applications than clients, their prefetchers are tuned using server benchmarks.

Eventually, the data for a prefetch request sent to the L2 cache/bus arrives. The line can be placed into the L1 data cache or not depending on a configuration parameter. If the configuration is set to drop the line, but the line is hit by a demand request before being dropped, the line is placed into the cache.

Prefetch Monitor

One possible challenge in employing eight prefetchers in one dual-core processor is the chance they might use up valuable bandwidth needed for demand load operations of running programs. To avoid this, the Intel Core microarchitecture uses a prefetch monitor with multiple watermark mechanisms aimed at detecting traffic overload. In cases where certain thresholds are exceeded, the prefetchers are either stalled or throttled down, essentially reducing the amount of aggressiveness in which prefetching is pursued. The result is a good balance between being responsive to the program's needs while also capitalizing on unused bandwidth to reduce memory latency. Overall, Intel Core microarchitecture's smarter memory access and more advanced prefetch techniques keep the instruction pipeline and caches full with the right data and instructions for maximum efficiency.

Summary

Intel Smart Memory Access plays an important contributing role in the overall energy-efficient performance of Intel Core microarchitecture. Through memory disambiguation, Intel Core microarchitecture increases the efficiency of out-of-order processing by providing the execution cores with the built-in intelligence to speculatively load data for instructions that are about to execute before all previous store instructions are executed. Through advanced IP-based prefetchers, Intel Core microarchitecture successfully prefetches memory data before it's requested, placing this data in cache for "just-in-time" execution. By increasing the number of loads that occur from cache versus main memory, IP-based prefetching reduces memory latency and improves performance. Included with the other four major "ingredients"—Intel Wide Dynamic Execution, Intel Advanced Digital Media Boost, Intel Advanced Smart Cache, and Intel Intelligent Power Capability—of the Intel Core microarchitecture, Intel Smart Memory Access plays an important role in the microarchitecture's ability to deliver increased performance and performance-per-watt.

Author's Bio

Jack Doweck is an Intel principal engineer in the Mobility Group and is the lead architect of the new Intel Core microarchitecture that is the basis for the new Intel Xeon 5100 series of processors and the Intel Core 2 Duo processors. During the definition stage of the Intel Core microarchitecture, Doweck defined the Memory Cluster microarchitecture and cooperated in defining the Out of Order Cluster microarchitecture. Doweck holds eight patents and has six patents pending in the area of processor microarchitecture. He has received three Intel Achievement Awards. Doweck joined Intel's Israel Design Center in 1989. He received a B.S. in electrical engineering and an M.S. in computer engineering from the Technion-Israel Institute of Technology in Haifa, Israel.

Learn More

Find out more by visiting these Intel Web sites:

Intel Core Microarchitecture
www.intel.com/technology/architecture/coremicro

Intel Xeon 51xx Benchmark Details
www.intel.com/performance/server/xeon

Intel Multi-Core
www.intel.com/multi-core

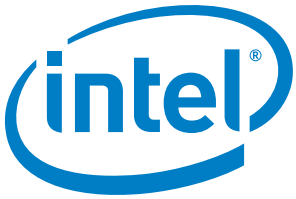
Intel Architectural Innovation
www.intel.com/technology/architecture

Energy-Efficient Performance
www.intel.com/technology/eep

References

[Bekerman] M. Bekerman, A. Yoaz, F. Gabbay, S. Jourdan, M. Kalaev, and R. Ronen, "Early Load Address Resolution via Register Tracking, in Proceedings of the 27th Annual International Symposium on Computer Architecture, pages 306-315, 2000"

[Wechsler] O. Wechsler, "Inside Intel® Core™ Microarchitecture: Setting New Standards for Energy-Efficient Performance," Technology@Intel Magazine, 2006.



www.intel.com

Printed in the United States. 0706/RMR/HBD/PDF 314175-001US