IBM POWER6 microarchitecture

H. Q. Le
W. J. Starke
J. S. Fields
F. P. O'Connell
D. Q. Nguyen
B. J. Ronchetti
W. M. Sauer
E. M. Schwarz
M. T. Vaden

This paper describes the implementation of the IBM POWER6™ microprocessor, a two-way simultaneous multithreaded (SMT) dual-core chip whose key features include binary compatibility with IBM POWER5™ microprocessor-based systems; increased functional capabilities, such as decimal floating-point and vector multimedia extensions; significant reliability, availability, and serviceability enhancements; and robust scalability with up to 64 physical processors. Based on a new industry-leading high-frequency core architecture with enhanced SMT and driven by a high-throughput symmetric multiprocessing (SMP) cache and memory subsystem, the POWER6 chip achieves a significant performance boost compared with its predecessor, the POWER5 chip. Key extensions to the coherence protocol enable POWER6 microprocessor-based systems to achieve better SMP scalability while enabling reductions in system packaging complexity and cost.

Introduction

IBM introduced POWER6* microprocessor-based systems in 2007. Based upon the proven simultaneous multithreaded (SMT) implementation and dual-core technology in the POWER5* chip [1], the design of the POWER6 microprocessor extends IBM leadership by introducing a high-frequency core design coupled with a cache hierarchy and memory subsystem specifically tuned for the ultrahigh-frequency multithreaded cores.

The POWER6 processor implements the 64-bit IBM Power Architecture* technology. Each POWER6 chip (**Figure 1**) incorporates two ultrahigh-frequency dual-threaded SMT processor cores, a private 4-MB level 2 cache (L2) for each processor, a 32-MB L3 cache controller shared by the two processors, two integrated memory controllers, an integrated I/O controller, an integrated symmetric multiprocessor (SMP) coherence and data interconnect switch, and support logic for dynamic power management, dynamic configuration and recovery, and system monitoring. The SMP switch enables scalable connectivity for up to 32 POWER6 chips for a 64-way SMP.

The ultrahigh-frequency core represents a significant change from prior designs. Driven by the latency and throughput requirements of the new core, the large, private L2 caches represent a departure from the designs of the POWER4* [2] and POWER5 [1] processors, which employed a smaller, shared L2 cache. The large, victim L3 cache, shared by both cores on the chip and accessed in parallel with the L2 caches, is similar in principle to the POWER5 L3 cache, despite differences in the underlying implementation resulting from the private L2 caches. Likewise, the integrated memory and I/O controllers are similar in principle to their POWER5 counterparts. The SMP interconnect fabric and associated logical system topology represent broad changes brought on by the need to enable improved reliability, availability, and serviceability (RAS), virtualization [3], and dynamic configuration capabilities. The enhanced coherence protocol facilitates robust scalability while enabling improved system packaging economics.

In this paper, we focus on the microarchitecture and its impact on performance, power, system organization, and cost. We begin with an overview of the key features of the POWER6 chip, followed by detailed descriptions of the ultrahigh-frequency core, the cache hierarchy, the memory and I/O subsystems, the SMP interconnect, and the advanced data prefetch capability. Next, we describe how the POWER6 chipset can be employed in diverse system organizations.

High-frequency core design

The POWER6 core is a high-frequency design that is optimized for performance for the server market as well as power. It provides additional enterprise functions and RAS characteristics that approach mainframe offerings.

©**Copyright** 2007 by International Business Machines Corporation. Copying in printed form for private use is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract, but no other portions, of this paper may be copied or distributed royalty free without further permission by computer-based and other information-service systems. Permission to *republish* any other portion of this paper must be obtained from the Editor.

0018-8646/07/$5.00 © 2007 IBM

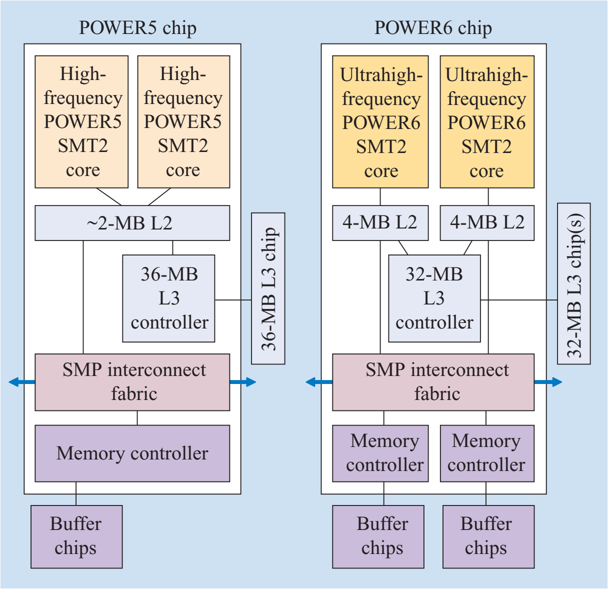

Figure 1

Evolution of the POWER6 chip structure. (SMT2: a dual-threaded simultaneous multithread.)

Its 13-FO4[1] pipeline structure yields a core whose frequency is two times that of the 23-FO4 POWER5 core. The function in each pipeline stage is tuned to minimize excessive circuitry, which causes delay and consumes excessive power. Speculation, which is costly at high frequency, is minimized to prevent wasted power dissipation. As a result, register renaming and massive out-of-order execution as implemented in the POWER4 [2, 4] and POWER5 [1] processor designs are not employed. The internal core pipeline, which begins with instruction fetching from the instruction cache (I-cache) through instruction dispatch and execution, is kept as short as possible. The instruction decode function, which consumed three pipe stages in the POWER5 processor design, is moved to the pre-decode stages before instructions are written into the I-cache. Delay stages are added to reduce the latency between dependent instructions. Execution latencies are kept as low as possible while cache capacities and associativity are increased. The POWER6 core has twice the cache capacity of its predecessor, providing one-cycle back-to-back fixed-point (FX) execution on dependent instructions, a two-cycle load for FX instructions, and a six-cycle floating-point (FP) execution pipe. The number of pipeline stages of the POWER6 processor design (from instruction fetch to an execution that produces a result) is similar to the POWER5 processor stages, yet the POWER6 core operates at twice the frequency of the POWER5 core.

In place of speculative out-of-order execution that requires costly circuit renaming, the POWER6 processor design concentrates on providing data prefetch. Limited out-of-order execution is implemented for FP instructions.

Dispatch and completion bandwidth for SMT has been improved. The POWER6 core can dispatch and complete up to seven instructions from both threads simultaneously. The bandwidth improvement, the increased cache capacity, cache associativity, and other innovations allow the POWER6 core to deliver better SMT speedup than the POWER5 processor-based system.

Power management was implemented throughout the core, allowing a clock gating efficiency[2] of better than 50%.

Balanced system throughput

While the frequency trade-offs were appropriate for the core, it did not make sense to extend ultrahigh frequency to the cache hierarchy, SMP interconnect, memory subsystem, and I/O subsystem. In the POWER5 processor design, the L2 cache operates at core frequency, and the remaining components at half that frequency. Preserving this ratio with the higher relative frequency of the POWER6 core would not improve performance but would actually impair it, since many latency penalties outside the core are more tied to wire distance than device speeds. Because the latencies in absolute time tend to remain constant, incorporating a higher-frequency clock results in added pipeline stages. Given that some time is lost every cycle because of clocking overhead, the net effect is to increase total latency in absolute time while increasing the power dissipation due to the increase in pipeline stages.

Therefore, for the POWER6 processor design, the L2 cache, SMP interconnect, and parts of the memory and I/O subsystems operate at half the core frequency, while the L3 cache operates at one-quarter, and part of the memory controller operates at up to 3.2 GHz. With lower power and slower devices, chip power is reduced. Because of their lower speed relative to the core, these components must overcome latency and bandwidth challenges to meet the balanced system performance requirements.

To achieve a balanced system design, all major subsystems must realize similar throughput improvements, not merely the cores. The cache hierarchy, SMP interconnect fabric, memory subsystem, and I/O subsystem must keep up with the demands for data generated by the more-powerful cores. Therefore, for the POWER6 processor design, the internal data throughput

[1] *FO4*, or fanout of 4, is the equivalent delay of an inverter driving four typical loads. FO4 is used to measure the amount of logic implemented in a cycle independent of technology.

[2] Percent of latches being gated off while running a typical workload.

Table 1 POWER5 processor to POWER6 processor throughput comparison (relative to core cycles).

Throughput resource	*POWER5 processor*	*POWER6 processor*
Data from L2 to core	32 B per cycle to data cache	64 B per 2 cycles
Data from core to L2	8 B per cycle	16 B per 2 cycles
L2 busy time	2 cycles for 64 B	4 cycles for 128 B
Aggregate L2 bandwidth	64 B × 3 per 2 cycles (2 cores)	128 B per 4 cycles (1 core)
Data from L3 to core/L2	128 B per 16 cycles	128 B per 16 cycles
Data from L2 to L3	128 B per 16 cycles	128 B per 16 cycles
Memory data into chip	16 B per (2 × 533 MHz) peak	16 B per (4 × 800 MHz) peak
Chip data out to memory	8 B per (2 × 533 MHz) peak	8 B per (4 × 800 MHz) peak
SMP fabric data into chip	48 B per 2 cycles	67% of 40 B per 2 cycles
Chip data out to SMP fabric	48 B per 2 cycles	67% of 40 B per 2 cycles

was increased commensurately with the increase in processing power, as shown in **Table 1**.

Since the L2 cache was designed to operate at half the frequency of the core, the width of the load and store interfaces was doubled; instead of driving 32 bytes of data per core cycle into the core, the POWER6 processor L2 drives an aggregate of 64 bytes every other core cycle. While the POWER5 processor L2 can obtain higher peak bandwidth when simultaneously delivering data to the data cache (D-cache) and I-cache, in realistic situations the D-cache and I-cache do not drive high-throughput requirements concurrently. This is because high bus utilization due to D-cache misses typically occurs in highly tuned single-threaded scenarios when there are multiple outstanding load instructions continuously in the pipeline, while I-cache misses interrupt the flow of instructions into the pipeline.

Instead of accepting 8 bytes of store data per core cycle, the POWER6 processor L2 accepts 16 bytes of store data every other core cycle. Note that the aggregate bandwidth of the POWER6 processor L2 per core per cycle is two-thirds that of the POWER5 processor L2. It does not have to scale perfectly for the following reasons: The POWER6 core has larger L1 caches, so there are fewer L1 misses driving fetch traffic to the L2; the POWER6 processor L2 can manage store traffic with 32-byte granularity, as opposed to 64-byte granularity for the POWER5 processor L2, so normally there is less L2 bandwidth expended per store. In addition, the POWER6 processor L2 is much larger per core than the POWER5 processor L2, so there are fewer L2 misses, driving fewer castout reads and allocate writes. (The term *castout* refers to the movement of deallocated, modified data from a given level in the cache hierarchy either to the next level of cache or to memory.)

For the POWER6 chip, the IBM Elastic Interface (EI) logic, which is used to connect to off-chip L3 cache data chips, I/O bridge chips, and SMP connections to other POWER6 chips, was accelerated to operate at one-half of the core frequency, keeping pace with corresponding interfaces in prior designs by achieving significantly higher frequency targets. The POWER6 processor L3 cache can read up to 16 bytes and simultaneously write up to 16 bytes every other core cycle, just as the POWER5 processor L3.

The POWER6 processor off-chip SMP interconnect comprises five sets of links. The organization of these is described later in the section "SMP interconnect." Each set can import up to 8 bytes and simultaneously export up to 8 bytes of data or coherence information every other core cycle. While this does not match the POWER5 processor SMP interconnect bandwidth per core cycle as seen by a given chip, the difference in system topology (described later in the section "SMP interconnect") and an increased focus on hypervisor and operating system optimizations for scalability drive a relaxation for the demand for interconnect data bandwidth.

The EI logic used for connectivity to off-chip memory buffer chips was accelerated to operate at 3.2 GHz when interacting with 800-MHz DRAM (dynamic random access memory) technology. By using both integrated memory controllers, a single POWER6 chip can read up to 16 bytes of data and simultaneously write up to 8 bytes of data or commands at 3.2 GHz. The I/O controller can read up to 4 bytes and simultaneously write up to 4 bytes of data to an I/O bridge chip every other core cycle.

Coherence protocol innovations to improve scalability

The coherence protocol and structures for POWER6 processor-based systems are based upon those found in

Table 2 POWER5 processor cache states.

State	*Description*	*Authority*	*Sharers*	*Castout*	*Source data*
I	Invalid	None	N/A	N/A	N/A
ID	Deleted, do not allocate	None	N/A	N/A	N/A
S	Shared	Read	Yes	No	No
SL	Shared, local data source	Read	Yes	No	At request
T	Formerly MU, now shared	Update	Yes	Yes	If notification
TE	Formerly ME, now shared	Update	Yes	No	If notification
M	Modified, avoid sharing	Update	No	Yes	At request
ME	Exclusive	Update	No	No	At request
MU	Modified, bias toward sharing	Update	No	Yes	At request

POWER5 processor-based systems. The strength of the POWER5 processor protocol lies in the combination of the weakly ordered IBM PowerPC* processor storage architecture, high-bandwidth SMP interfaces, low-latency intervention, and optimized locking primitives.

POWER5 processor-based systems have a nonblocking, broadcast-based coherence protocol provided by a robust set of request, response, and notification types. Coherence requests are broadcast to such units as caches as well as memory and I/O controllers. Upon snooping the requests, the units provide responses, which are aggregated to form the basis for a coherence transfer decision. The notification of the decision is subsequently broadcast to the units and indicates the final action to be taken.

The power of the protocol lies in its distributed management, facilitated in the POWER5 processor caches by nine associated cache states. **Table 2** provides details of these states.

With a broadcast-based snooping protocol such as that found in the POWER5 processor, coherence traffic and the associated bandwidth required grow proportionally with the square of the system size. As system-packaging cost implications of this bandwidth become more important, alternatives to globally snooped, broadcast-based protocols become more attractive. Approaches such as directory-based NUMA (nonuniform memory access) schemes have become popular [5] because they localize broadcasts to small nodes with directories that indicate when regions of memory owned by a given node are checked out to other nodes. This can greatly restrict traffic flow outside the node.

While such approaches can limit coherence traffic, they place a strong dependence upon operating system and hypervisor schedulers to keep processors and the memory they use localized within the small nodes. Operations with multinode scope incur significant latency degradation. Also, the directories can demand large chip-area budgets and may even reside on separate, additional hub chips, driving complexity into the design and often requiring additional software support to operate properly.

For some classes of POWER6 processor-based systems, it is important to retain the robust, large SMP scalability enjoyed by large POWER5 processor-based systems. Typical applications perform well on these systems but perform poorly on traditional directory-based NUMA systems. The differentiation provides great value in these cases and commands a premium price. For other classes of POWER6 processor-based systems, typical applications do not benefit from such differentiation. These systems compete directly with directory-based NUMA systems built from commodity parts, so it is critical to maintain the lowest possible cost structure. For POWER6 technology, it was necessary to develop a single design that incorporates a robust, global-broadcast-based protocol while also integrating a capability styled after directory-based NUMA, but with reduced power and area overhead and better latency.

Significant innovations have been incorporated into the coherence protocol to address this challenge. In addition to the globally broadcast request, response, and notification transport, with its distributed management using specialized cache states, a localized (or scope-limited) broadcast transport mechanism is also integrated. Thus, a given request can be broadcast globally or locally.

To enable the protocol, a new set of responses and notifications was added to provide more information for these local broadcasts. If it can be determined that all information necessary to resolve a coherence request exists within the local broadcast scope, then no global broadcast is necessary. If no such determination can be made, the request must be broadcast globally. To ensure a reasonable likelihood of a successful local resolution,

Table 3 POWER6 processor cache states added for multiscope coherence protocol.

State	*Description*	*Authority*	*Sharers*	*Castout*	*Source data*
IG	Invalid, cached scope-state	N/A	N/A	N/A	N/A
IN	Invalid, scope predictor	N/A	N/A	N/A	N/A
TN	Formerly MU, now shared	Update	Yes	Yes	If notification
TEN	Formerly ME, now shared	Update	Yes	No	If notification

Table 4 POWER6 processor cache states and scope-state implications.

State	*Implied scope state*	*Scope-state castout*
I	None	None
ID	None	None
S	Unknown	None
SL	Unknown	None
T	Shared copies probably global	Required, global
TE	Shared copies probably global	Required, global
M	Local	Optional, local
ME	Local	None
MU	Local	Optional, local
IG	Existing copies probably global	Required, global
IN	Existing copies probably local	None
TN	Shared copies all local	Optional, local
TEN	Shared copies all local	None

a mechanism similar to a NUMA directory is necessary. Instead of a new directory structure, the protocol introduces a new scope-state bit per 128-byte cache line in memory and a set of new cache states for the L2 and L3 caches (**Table 3**).

The scope-state bit in memory is integrated into the redundant content for error correction already stored in memory, so no cost is added. For each 128-byte cache line, the bit indicates whether the line might be in use outside of the local scope where the memory resides. Since it is stored with the data bits, the state bit is automatically read or written whenever the data is read or written. The four new cache states provide a means of caching the scope-state bit in the L2 and L3 caches, either by itself or along with the data it covers. As shown in **Table 4**, a number of the original nine cache states and the four new ones provide low-latency access to high-usage scope state while protecting memory from increased traffic related to scope-state queries and updates. Note that when cached scope state is deallocated, it is typically cast out (i.e., written back) to memory. For cases in which the implied scope state might be global, the castout is functionally required to ensure that coherence is maintained. For cases in which the implied scope state is known to be local, the castout is optional, as it is desirable but not necessary to localize the broadcast scope for subsequent operations.

The combination of the scope-state bit in memory and the four new cache states provides a low-cost alternative to a NUMA directory and integrates cleanly into the nonblocking-broadcast distributed-coherence protocol. As some workloads localize well and others do not, the design of the POWER6 processor incorporates a number of predictors to determine whether a given coherence request should make a local attempt or immediately broadcast globally. For workloads that exhibit a high degree of processor-to-memory localization, and for workloads that have varying mixtures of locally resolvable traffic, laboratory results show that scope-limited speculative snoop resolution is highly effective.

POWER6 chip physical overview

The POWER6 chip is manufactured using the IBM CMOS (complimentary metal oxide semiconductor) 11S 65-nm silicon-on-insulator (SOI) copper process, which incorporates SOI devices and a ten-level copper interconnect. The chip size is 341 mm^2. It utilizes 790 million transistors and contains 1,953 signal and test I/Os and 5,399 power and ground I/Os. A comparison between POWER6 and POWER5 processor signal I/O counts by major functional group is shown in **Table 5**.

The POWER6 chip floorplan (**Figure 2**) illustrates the balanced system design. The layout and placement of the cores, L2 caches, interconnect buses, and memory controllers balance and complement each other, as described below.

A key challenge for the POWER6 processor design was dealing with the thermal issues associated with an ultrahigh-frequency, dual-core chip. The physical separation of the cores provides improved thermal properties by distributing the hottest regions (the L1 D-caches found within each core) to opposite ends of the chip to enable more effective heat dissipation.

Another challenge for the POWER6 processor design was scaling the latency (as measured in core cycles) to the

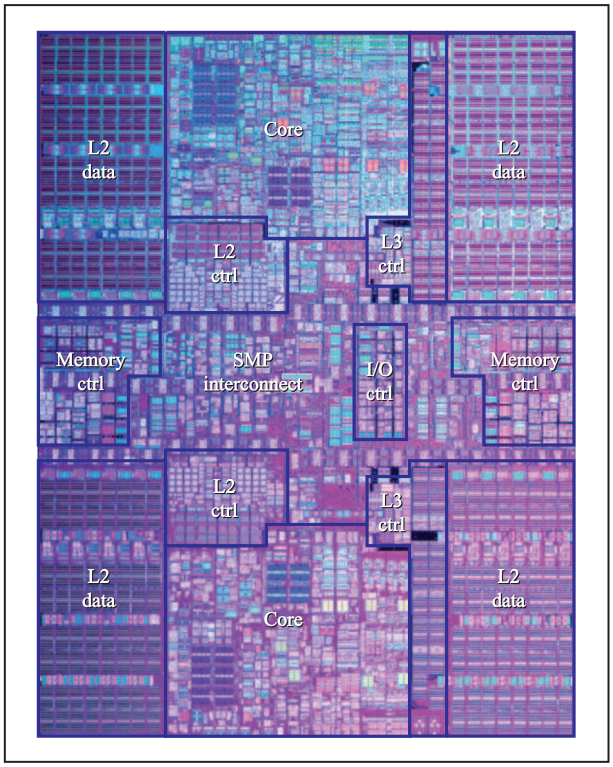

Figure 2

Die photograph of the POWER6 chip. (ctrl: controller.)

Table 5 POWER5 processor to POWER6 processor functional signal I/O comparison.

Functional I/O group	*POWER5 processor* (I/Os)	*POWER6 processor* (I/Os)
Memory interfaces	∼410	∼400
SMP fabric interfaces	∼1,440	∼900
L3 cache interfaces	∼360	∼420
I/O subsystem interfaces	∼100	∼100

L2 cache, as the frequency was doubled from that of the previous generation. Designing an architecture for a private L2 cache for each core (a departure from the shared L2 cache found in the designs of the POWER4 and POWER5 processors) enabled a number of optimizations in the physical layout of the POWER6 chip to address L2 latency and bandwidth.

Figure 2 shows what we call the *butterfly* layout of the private L2 cache data arrays, with half located on the left side of the core and the other half located on the right side. This yields better L2 latency by reducing the physical distance from the furthest data arrays to the L1 D-cache inside the core. Additionally, it doubles the combined physical width of the data bus that brings data from the L2 cache into the core. This means that twice the wiring resource can be committed to this critical task, enabling the use of larger, faster wires to provide further improvements in L2 latency while increasing the bandwidth, as described earlier in the section "Balanced system throughput."

As can be seen in Figure 2, each core, the associated L2 cache controller, and the left-side set of L2 cache data arrays are arranged in what we call the *latency triangle* in order to provide the shortest physical roundtrip distance from the generation of an L2 fetch request to the return of L2 cache data into the core.

Together, these properties allow the L2 cache capacity to increase from less than 1 MB per core in the POWER5 chip to 4 MB per core in the POWER6 chip while significantly reducing the latency in absolute time, thereby minimizing the impact of the L2 latency growth in core cycles. The net effect of changing from shared L2 caches to private L2 caches was to improve latency and increase per-core capacity while eliminating any impact from multicore competition for capacity.

By placing the data interfaces at the inward-facing edge of the L2 cache data arrays and the coherence interfaces at the inward-facing edge of the L2 and L3 cache controllers, it was possible to localize the high-bandwidth interconnect fabric within a central horizontal region of the POWER6 chip. Surrounding this region, the symmetric layout of cores and associated L2 caches (at the top and bottom of the chip) and the dual memory controllers (at the left and right edges) results in a floorplan with solid thermal properties. It also offers balanced internal bandwidth that utilizes premium wiring resources to improve latencies and off-chip I/O placements well optimized for module and system packaging.

Processor core

The POWER6 core microarchitecture was developed to minimize logic content in a pipeline stage. Circuit area and speculative work are minimized in order to reduce wasted power dissipation. The result is a 13-FO4 design with short pipeline, large split L1 instruction, and D-caches supporting two-way SMT. Additional virtualization functions, decimal arithmetic, and vector multimedia arithmetic were added. Checkpoint retry and processor sparing were implemented.

Instruction fetching and branch handling are performed in the instruction fetch pipe (**Figure 3**). Instructions from the L2 cache are decoded in precode stages P1 through P4 before they are written into the L1 I-cache. Branch prediction is performed using a

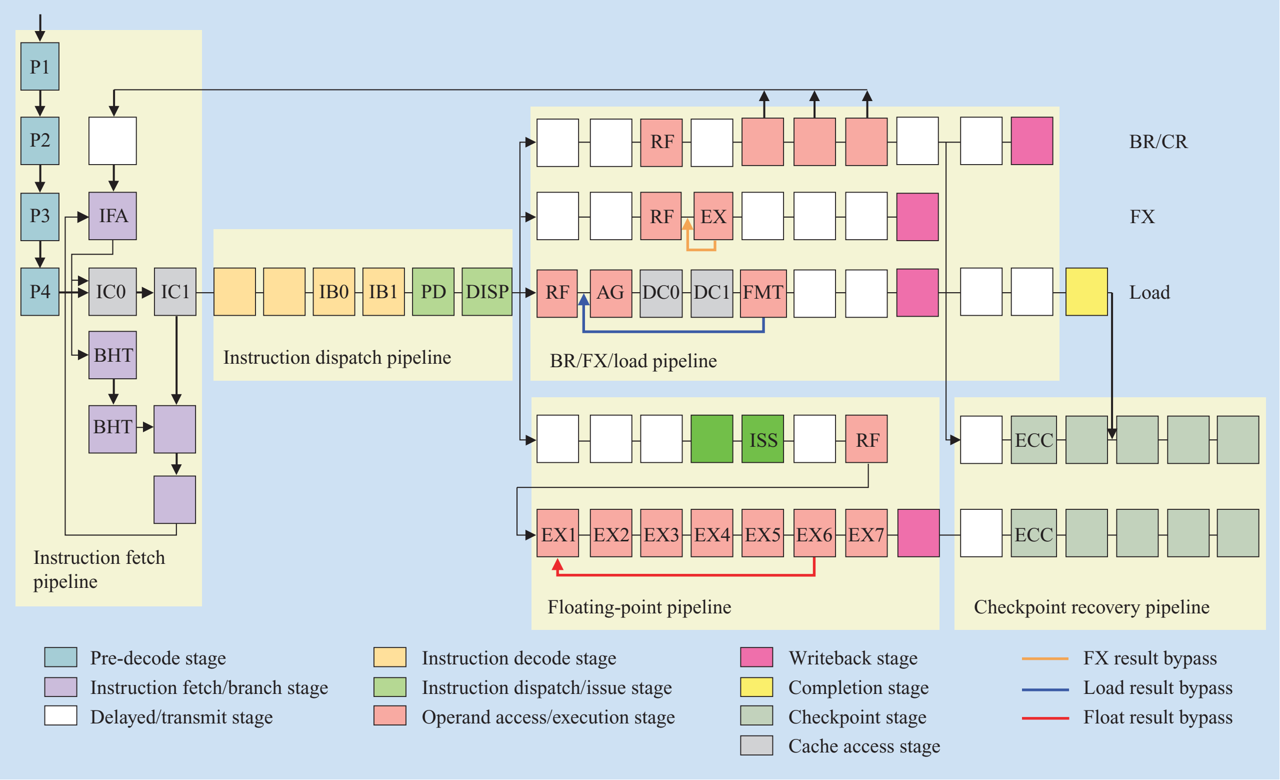

Figure 3

POWER6 core pipeline. (AG: address generation; BHT: branch table access and predict; BR: branch; DC: data-cache access; DISP: dispatch; ECC: error-correction code; EX: execute; FMT: formatting; IB: instruction buffer; IC0/IC1: instruction-cache access; IFA: instruction fetch address; ISS: issue; P1–P4: pre-decode; PD: post-decode; RF: register file access.)

16K-entry branch history table (BHT) that contains 2 bits to indicate the direction of the branch. Up to eight instructions are then fetched from the L1 I-cache and sent through the instruction decode pipeline, which contains a 64-entry instruction buffer (I-buffer) for each thread. Instructions from both threads are merged into a dispatch group of up to seven instructions and sent to the appropriate execution units.

Branch and logical condition instructions are executed in the branch and conditional pipeline, FX instructions are executed in the FX pipeline, load/store instructions are executed in the load pipeline, FP instructions are executed in the FP pipeline, and decimal and vector multimedia extension instructions are issued through the FP issue queue (FPQ) and are executed in the decimal and vector multimedia execution unit. Data generated by the execution units is staged through the checkpoint recovery (CR) pipeline and saved in an error-correction code (ECC)-protected buffer for recovery. The FX unit (FXU) is designed to execute dependent instructions back to back.

The FP execution pipe is six stages deep. **Figure 4** shows the pipeline for both the POWER6 and the POWER5 design with cycle-time delay for each stage, starting with instruction fetching from the I-cache to the time the result is available for subsequent instruction. FX load/store instructions are executed in order with respect to each other, while FP instructions are decoupled from the rest of the other instructions and allowed to execute while overlapping with subsequent load and FX instructions. Additional emphasis was put in the design to minimize the memory effect: prefetching to multiple levels of caches, speculative execution of instructions to prefetch data into the L1 D-cache, speculative prefetching of instructions into the L1 I-cache, providing a load-data buffer for FP load instructions, hardware stride prefetching, and software-directed prefetching. Buffered stages are added between the dispatch stage and the execution stage to optimize certain execution latency between categories of instructions. The following are examples:

- The FX instructions are staged for two additional cycles prior to execution in order to achieve a one-

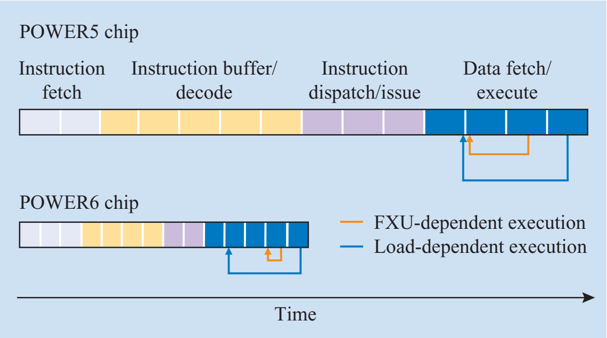

Figure 4

Internal POWER6 processor pipeline compared with the POWER5 processor pipeline with cycle time.

cycle load-to-use between a load instruction and a dependent FX instruction.

- Branch instructions are staged for two additional cycles to line up with FX staging instructions in order to avoid an additional branch penalty on an incorrect guess.
- FP instructions are staged through the FPQ for six cycles to achieve a zero-cycle load-to-use instruction between a load instruction and the dependent FP instruction.

The POWER6 core contains several units. The instruction fetch unit (IFU) performs instruction fetching, instruction pre-decoding, branch prediction, and branch execution. The instruction dispatch unit (IDU) performs instruction dispatch, issuing, and interrupt handling. The two FXUs, the two binary FP units (BFUs), the decimal FP unit (DFU), and the vector multimedia extension (VMX) unit are responsible for executing the corresponding set of FX, FP, decimal, and VMX instructions. The two load/store units (LSUs) perform data fetching. The recovery unit (RU) contains the data representing the state of the processor that is protected by ECC so that the state of the processor can be restored when an error condition is detected.

Instruction fetching and branch handling

The POWER6 core contains a dedicated 64-KB four-way set-associative L1 I-cache. Fast address translation is performed using a 64-entry instruction effective-to-real address translation (I-ERAT) table, which supports 4-KB and 64-KB page sizes. Larger pages (16 MB and 16 GB) are mapped into multiple 64-KB entries in the I-ERAT table.

The IFU fetches instructions from the L2 cache into the L1 I-cache. These fetches are either demand fetches that result from I-cache misses or prefetches that result from fetching up to two sequential cache lines after a demand fetch. Fetch requests are made for a cache line (128 bytes), and data is returned as four sectors of 32 bytes each. Demand and prefetch requests are made for both instruction threads, and data may return in any order, including interleaving of sectors for different cache lines. Up to 32 instruction fetch requests can be initiated from the core to the L2 cache. Instructions are fetched from the I-cache and written into the I-buffer in the IDU.

One major difference between the design of the POWER6 and that of the POWER4 and POWER5 processors [1, 2] is to push many of the decode and group formation functions into pre-decode and thus out of the critical path from the I-cache to the instruction dispatch. The POWER6 processor also recodes some of the instructions in the pre-decode stages to help optimize the implementation of later pipeline stages. The POWER6 processor requires four cycles of pre-decode and group formation before writing instructions into the I-cache, thus increasing the L2 access latency. The POWER6 processor still accomplishes a significant performance improvement by removing these functions from the much more critical I-cache access path. (Performance data shows a loss of less than 0.5% for every stage added to the L2 instruction access latency, compared with a loss of about 1% for each added stage after the I-cache access.)

The IFU performs the following major functions:

Instruction recoding—Instructions can be recoded as part of the pre-decode function and before writing them into the I-cache. An example of useful recoding is switching register fields in some class of instructions in order to make the instructions appear more consistent to the execution units and save complexity and multiplexers in critical paths.

The most extensive instruction recoding on the POWER6 processor occurs for branches. Relative branches contain an immediate field that specifies an offset of the current program counter as a target address for the branch. Pre-decode adds the immediate field in the branch to the low-order bits of the program counter, thus eliminating the need for an adder in the critical branch execution path. Pre-decode also determines whether the high-order bits of the program counter have to be incremented, decremented, or used as is to compute the branch target and that information is encoded in the recoded branch. Absolute branches can also be handled by this approach by signaling to use the sign-extended offset as a branch target. Information about conditional branch execution is also encoded in the branch to enable branch execution to achieve the cycle-time target for this type of critical instruction.

Instruction grouping—Instructions are grouped in the pre-decode stages and the group information is stored in the L1 I-cache. Grouping allows ease of dispatch determination in the critical dispatch stage. Only instructions of the same thread are allowed in the same group. In general, the number of instructions in the group matches the dispatch bandwidth (five instructions per thread) as well as the execution resources. Except for FP instructions, instructions within a group are independent of one another. A taken branch terminates a group.

Each instruction in the I-cache carries a start bit that controls dispatch group formation in the IDU. If the start bit associated with an instruction is set, that instruction starts a new group. Otherwise, it is part of a larger group of instructions that started earlier in the instruction stream. The POWER6 processor dispatch groups contain, at most, five instructions. An extensive set of rules from very simple to rather complex ones determine whether the start bit of a particular instruction is set on the basis of the instruction itself and the instructions immediately preceding it.

In order to use previous instruction characteristics to decide on the setting of a particular start bit, the information about the immediately preceding instructions must be available. If the preceding instructions are within the cache sector being pre-decoded, the information is readily available. If the preceding instructions are in a preceding sector (e.g., the first instruction of one sector has all of its immediately preceding instructions in another sector), the POWER6 processor pre-decode maintains information for the last instructions in sectors that have recently been fetched. This information is maintained for each thread for the demand cache line and two prefetched cache lines. Let us assume the first sector of a demand request for thread 0 is received from the L2 cache. Pre-decode stores the information about the last four instructions in that sector (no more than four are required, as there are at most five instructions in a group). When the second sector is received, the stored information is used by pre-decode to form correct groups across sector boundaries. There are two major restrictions to this process. The first restriction is that the first instruction in the first sector of a cache line is always first in a group because no information is maintained across cache lines. The second major restriction is that the L2 cache tries to return the sectors of a line in order, but in rare cases sectors may be returned in a different order. In cases in which sectors return nonsequentially, the stored information is useless, and the first instruction in the sector will start a group.

Although start bits are normally set statically and written into the I-cache, there are a few exceptions in which start bits are set dynamically after reading the instruction (and the static start bit) from the I-cache. Most notably, this happens for an instruction that is the target of a taken branch. The next instruction after a taken branch always starts a group because no information is available about the previous instructions across a taken branch. There are a few cases in which a start bit is forced—on the first instruction of a sector fetched after certain events in the IFU—to avoid potential problems and simplify controls.

A sample list of rules used to form instruction groups follows. (Other, more specific rules also exist, such as restrictions for dependencies between the first and last instruction of a five-instruction group to simplify the pre-decode logic.) A rule states what is not allowed to occur within a group; consequently, the first instruction that would violate a rule starts a new group.

- A group cannot use more resources than are available in the POWER6 processor. Available resources are two FXUs, two LSUs, two FP or VMX units, and one branch unit.
- A group cannot have a write-after-write dependency on any general-purpose register (GPR), FP register (FPR), CR field, or the FX exception register (XER).
- A group cannot have a read-after-write dependency on the XER.
- A group cannot have a read-after-write dependency on a GPR, except for an FX instruction or an FX load, followed by an FX store of the same register.
- A group cannot have a read-after-write dependency on a CR field except for an FX instruction setting the CR field, followed by a conditional branch dependent on the CR field. Thus, the most important combination of a compare followed by a dependent conditional branch can be in one group.
- A group cannot have an FP multiply–add (FMA) followed by an instruction reading the target FPR of the FMA.
- A group cannot have an FP load followed by an FP store of the same FPR.

Branch execution—The POWER6 processor can predict up to eight branches in a cycle (up to the first taken branch in a fetch group of up to eight instructions). Prediction is made using a 16K-entry, 2-bit BHT; 8-entry, fully associative count cache; and a 6-entry link stack. The BHT and the count cache are shared between the two threads. Up to ten taken branches per thread are allowed to be pending in the machine.

The POWER6 processor is optimized to execute branches as early as possible and as fast as possible. The pre-decode assists by precomputing the branch target and providing encoded information for the evaluation of conditional branches. Another unique design technique is allowing conditional branch execution to occur in any of three stages in the POWER6 processor execution pipeline, whereas instruction execution normally occurs in a fixed pipeline stage. The stage at which the branch is actually evaluated is determined dynamically when the branch is dispatched to the branch execution unit. This stage depends on the availability of the CR bits. Because compare and dependent branches are allowed to be in the same dispatch group, they may be dispatched in the same cycle. In that case, the compare evaluates the CR field late with respect to branch dispatch, and the branch execution must wait for the result of the compare instruction. If the condition can be evaluated earlier, the branch can be evaluated earlier as well. If the compare dispatches one cycle earlier than the branch, the branch execution is performed a cycle earlier. If the dispatch distance is two or more cycles, the branch execution is performed two cycles earlier. When the compiler can schedule the compare instruction earlier, this implementation reduces the branch penalty for incorrect guesses by up to two cycles.

Another special feature is the ability to hold instruction fetch for unpredictable branch-to-count instructions, wait for the FX execution unit to deliver the correct count register (CTR) value, and use that value directly to fetch the next instructions without waiting for branch execution. This improvement is new in the POWER6 processor design and is more valuable here because of the relatively small count cache. This type of poorly predicted branch is commonly used for branches beyond the horizon of relative branches, dynamic method calls in Java** code, and as computed branches, using a table for switch statements.

The count cache logic maintains a confirm bit and an unpredictable bit for every entry. When a new entry is written into the count cache, it starts with the confirm bit set to 1 and the unpredictable bit set to 0. An incorrect prediction will set the confirm bit to 0. An incorrect prediction with the confirm bit already set to 0 will set the unpredictable bit. A given branch-to-count instruction (bcctr) is considered unpredictable by the fetch logic either if it does not match in the count cache or if there is a match but the unpredictable bit is 1. An unpredictable bcctr will cause the fetch logic to stop fetching and wait for the execution of a move-to-CTR (mtctr) instruction, which will put the correct CTR value on the FXU bus. The fetch logic uses that value as the bcctr target address.

There are a couple of special cases to be considered for this mechanism. The branch execution unit contains a state machine to keep track of bcctr and mtctr instructions and cause the correct action depending on the instruction sequence. One common problem is that a bcctr is considered unpredictable causing fetch to stop, but there is no mtctr instruction to set the CTR (which could have been set much earlier). The branch execution state machine detects this condition, activates the fetch logic, and provides the correct fetch address. Another interesting, though somewhat deleterious, situation occurs if there are two mtctr instructions before the bcctr. The fetch logic picks up the first CTR value, but the second one is actually the correct target address. The branch execution state machine handles this case by forcing the bcctr to redirect to the correct address when the bcctr executes. These special cases are unlikely to occur in normal code. The mechanism covers the vast majority of performance-relevant cases and improves bcctr performance significantly.

Instruction sequencing

Instruction dispatching, tracking, issuing, and completing are handled by the IDU. At the dispatch stage, all instructions from an instruction group of a thread are always dispatched together. Both threads can be dispatched simultaneously if all of the instructions from both threads do not exceed the total number of available execution units. The POWER6 processor can dispatch up to five instructions from each thread and up to seven instructions from both threads.

In order to achieve high dispatch bandwidth, the IDU employs two parallel instruction dataflow paths, one for each thread. Each thread has an I-buffer that can receive up to eight instructions per cycle from the I-cache. Both I-buffers are read at the same time, for a maximum of five instructions per thread. Instructions from each thread then flow to the next stage, in which the non-FP unit (FPU) and VMX dependency-tracking and resource-determination logic is located. If all of the dependencies for instructions in the group are resolved, then the instruction group is dispatched. Otherwise, the group is stalled at the dispatch stage until all of the dependencies are resolved.

Tracking of non-FPU and VMX instruction dependencies is performed by a target table, one per thread. The target table stores the information related to the whereabouts of a particular instruction in the execution pipe. As instructions flow from the I-buffer to the dispatch stage, the target information of those instructions is written into the target table. Subsequent FX instructions access the target table to obtain dependency data so that they can be dispatched appropriately. FPU and VMX instruction dependencies are tracked by the FPQ located downstream from the dispatch stage.

FPU and VMX arithmetic instructions are dispatched to the FPQ. Each FPQ can hold eight dispatch groups, and each group can have two FPU or VMX arithmetic instructions. The FPQ can issue up to two arithmetic FPU or VMX instructions per cycle. In order to achieve zero-cycle load-to-use for load floats feeding arithmetic FPU instructions, the FPU instructions are staged six cycles after the dispatch stage through the FPQ to line up with load data coming from the LSU. If the load data cannot be written into the FPR upon arrival at the FPU, it is written into a 32-entry load target buffer (16 per thread). The load target buffer allows up to 16 load instructions per thread to execute ahead of arithmetic FP instructions, thus eliminating the effect of the six-cycle FP pipe stages. Arithmetic FPU instructions can also be issued out of order with respect to other FPU instructions. At most, eight FPU instructions can be issued out of order from the FPQ.

In order to enhance performance, load lookahead (LLA) execution is employed to facilitate load prefetching to the L1 D-cache after a load miss or ERAT miss. When a miss occurs, the IDU will enter LLA mode, whereby instructions after the missed load continue to be dispatched from the I-buffer and executed. In LLA mode, the instructions can produce the result and pass it to the dependent instructions before the data reaches the writeback stage. These results are discarded once they pass the writeback stage, and any instructions that are dependent on these discarded results are no longer allowed to execute. Prefetch requests are initiated for load instructions that are successfully executed under LLA. When a thread enters LLA mode, its thread priority is switched to a lower priority in order to give non-LLA-mode threads higher dispatch priority. When data for the original load miss is returned from the L2, the original load and all of the subsequent instructions are read out of the I-buffer for redispatching. The thread dispatch priority will also revert to its original setting prior to the initiation of the LLA mode.

As a further LLA enhancement, in single-thread mode, execution results are not discarded as they pass through the writeback stage; instead, they are written into the GPR of the other thread and become available for subsequent instructions to resolve dependencies. This mechanism enables more load instructions to be executed for prefetching.

A completion table is employed by the IDU to track a high number of instructions in flight. Each thread uses a ten-entry completion table. Each completion table entry holds the information necessary to track a cache-line's worth of instructions (up to 32 sequential instructions). When a taken branch is detected in the fetching instruction stream, a new completion table entry is allocated for the instructions after the predicted taken branch. A new completion table entry is also allocated when an I-cache line is crossed. In effect, the completion table can track up to 320 instructions, or ten taken branches, per thread.

FX instruction execution

The POWER6 core implements two FXUs to handle FX instructions and generate addresses for the LSUs. The FXU executes most instructions in a single pipeline stage. One of the signature features of the POWER6 processor FXU is that it supports back-to-back execution of dependent instructions with no intervening cycles required to bypass the data to the dependent instruction. Several enhancements to the FXU logic and circuits relative to the previous implementations of the Power Architecture technology were employed to achieve this [6].

The floorplan is organized in two stacks, with the FX circuits in one stack and the load/store operand circuits in the other stack.

Along with the floor planning, the logic and circuits in the execute stage were optimized to reduce delay. Typically, an FXU uses an XOR circuit on one of the operands so that the input to the adder can be complemented for implementing subtract- and compare-type instructions. To reduce the number of circuits in the execute stage, the complement function of the adder was moved to the previous cycle. As a result, complement data or noncomplement data is bypassed to the dependent instruction on the basis of the type of instruction that processes the data. For example, for back-to-back dependent instructions in which the second instruction is a subtract instruction, the execution of the first instruction is modified such that the result is the ones complement of the expected result. This allows circuit stages to be removed from the critical path of the adder.

FX multiply and divide are executed by the FPU. Data is read from the GPR, and is sent to the BFU, and the results are sent back to the FXU to be written to the GPR. This implementation allows consecutive multiply instructions to be executed in a pipeline fashion at a rate of one multiply every two cycles.

Another structure that was modified from previous designs is the result multiplexer. More conventional implementations have a four- or five-input result multiplexer that brings together the results of several macros (such as adder, rotator, and logical) to form a single result bus, which can then be forwarded. The POWER6 processor FXU uses a single three-input NAND gate to perform this function by taking input from the adder, rotator, and logical operand macros. Results from all other functions are multiplexed in the logical operand macro or other pipeline stages. If a macro is not sourcing data, then it will drive 1s into its input of the three-input NAND.

The adder was speeded up by moving the bitwise propagate and generate functions into the previous cycle after all of the bypass multiplexing. The delay required to calculate the propagate and generate functions before the operand latch is less than the delay required if the function were implemented after the operand latch. This saves time in the adder functional implementation.

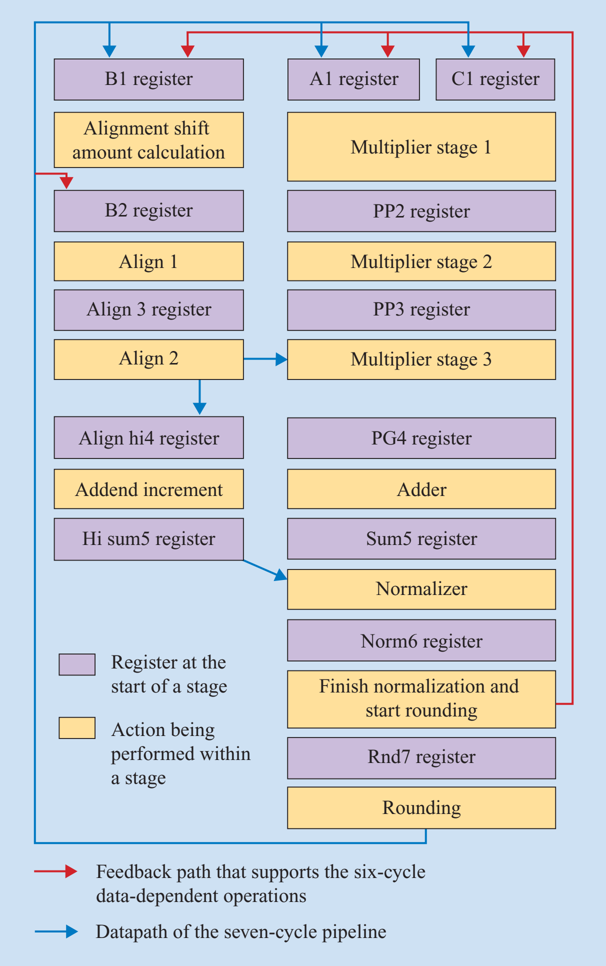

Figure 5

Binary floating-point pipeline.

The rotator was speeded up by removing the 2:1 multiplexer used to implement the most significant word of the operand for the Power Architecture technology 32-bit rotate and shift instructions. To compensate, each byte of the byte multiplexer in the rotator has separate controls.

The operand multiplexing and latching was speeded up by using a dynamic circuit for the last stage of the bypass multiplexer and by using a pulsed latch to stage the operand. In addition, the circuit tuning for the execute stage was done with all of the macros in the model. This allowed us to make trade-offs to speed up the most critical paths at the expense of less critical paths.

Another feature of the POWER6 processor FXU is the implementation of delayed stores. This enables better group utilization and reduces delay when stored data is dependent on the target of an older load instruction. Store instructions read GPRs to get the operands used to calculate the effective address (EA) and to get the data that is to be placed into memory. Store instructions read operands the cycle after dispatch but cannot write the data for several cycles until the address has been translated, the cache directory has been checked, and address conflicts have been resolved. This allows an opportunity for the FXU to capture the data for the store instruction as late as six cycles after dispatch. Consider the following sequence of instructions:

```
ld   r5, 0x4(r3)
std  r5, 0x4(r4)
```

Without the delay store feature, the store dispatch and its execution must stall for three cycles, waiting for the data from the load instruction. With the delay store feature, the dependent store instructions can be placed in the same group and are executed in the same cycle as the load instruction, thus yielding significant performance gain.

Binary FP instruction execution

The POWER6 core includes two BFUs, essentially mirrored copies, which have their register files next to each other to reduce wiring. The POWER6 processor binary FP design exceeds that of the POWER4 and POWER5 processors by maintaining the same number of cycles to execute an FP instruction even though the cycle time is much more aggressive. The POWER4 and POWER5 processor BFUs have a six-cycle pipeline that supports an independent multiply–add instruction every cycle and requires six cycles between dependent operations with a cycle time of approximately 23 FO4 [7, 8]. For the POWER6 processor design, the cycle time is much more aggressive, 13 FO4, but the infinite cache performance is maintained [6, 9, 10]. To offset some of the cycle-time differences, an additional pipeline stage is used, but it is designed so that it does not have an impact on performance. New bypass paths are added to make up for the additional latency. Rather than waiting for the result to be rounded, an intermediate result is fed back to the beginning of the pipeline prior to rounding and prior to normalization. Several performance enhancements, such as store folding and division and square root enhancements, are described along with the basic pipeline.

The pipeline is shown in **Figure 5**. The pipeline accepts a new fused multiply–add operation every cycle. The operation is $A \times C \pm B$, where A and C are the multiplicands and B is the addend. The dataflow supports operands up to 64 bits wide, which is especially useful for performing FX multiplication and provides additional guard bits for division algorithms. Data from the FPRs or from the FXU is fed into the three operand registers, A1, B1, and C1, where the number 1 indicates the first stage of the pipeline. The multiplier uses a radix-4 Booth algorithm to reduce the 64-bit × 64-bit multiplication. The multiplication takes three cycles to reduce 33 partial products (PPs) to 2, and then the addend is combined. In the fourth and in part of the fifth cycle, a 120-bit end-around carry (EAC) adder [11] is implemented to produce an only positive magnitude result. In the rest of the fifth cycle, two stages of the normalizer are implemented, which is completed in the sixth cycle with an 8:1 multiplexer. The rest of the sixth cycle involves starting the incrementation, and in the seventh cycle, the rounding is completed.

The latency of the pipeline is seven cycles, but it appears to be effectively six cycles for data-dependent operations. This is achieved by feeding back the data prior to rounding and prior to complete normalization. Rounding is corrected by adding terms in the counter tree. For instance, if the result is to be fed back into operand A, then $A = A' + 2^{-u}$, where 2^{-u} is the increment due to rounding. Then, $A \times C + B = (A' + 2^{-u}) \times C + B = A' \times C + B + C * 2^{-u}$.

Thus, an extra C has to be added, which must be shifted to the 1-ulp[3] position of A. Because A could be single precision or double precision, the correction term C could be located in two different places. Altogether there are six different rounding correction terms that could be added to correct for A, C, or both, and for single or double precision.

Because the design did not meet cycle-time objectives with this feedback path, it was modified to use an unrounded intermediate result and was also unnormalized. The normalizer uses leading-zero anticipation (LZA) logic to predict the shift amount within 1 bit. This requires a correction shift at the end of the normalizer, which is usually implemented with a 2:1 multiplexer in which the select signal is the most significant bit of the data. To repower this data bit and use it as a select signal in a 120-bit 2:1 multiplexer requires about 4.5 FO4, which is about half a cycle of available logic in a 13-FO4 cycle. To skip this stage of shifting and feedback, a partially normalized result requires an additional bit in the dataflow and a 1-bit mask function. The rounding correction term gets more complex since the least significant bit can now be in two different locations for each data format, which increases the multiplexer to 12 different possibilities. Thus, a significant savings in cycle time is possible by feeding back the partially normalized, unrounded, intermediate result, but at the cost of complexity.

The multiplier and multiplicand are corrected by adding the rounding correction term in the multiplier counter tree, but for the addend, there is no place to add in the correction. For the addend correction, the exponent is fed back after partial normalization and the significand is fed back a cycle later, after rounding to the B2 register. This is possible since the addend significand does not have any computation in cycle 1. Thus, a six-cycle feedback is possible to any of the input operands, or to even more than one.

Feedback paths are also important in the execution of a store instruction. There is an advantage to pipeline stores with FP operations, but they are typically dependent on a prior arithmetic instruction. Rather than wait the full pipeline depth to resolve the dependency, feedback paths are implemented at the bottom of the pipeline to bypass to a store. This is called *store folding*. To eliminate feedback paths to all stages of the pipeline, an additional read port in the FPR is implemented. The read port is used in a late cycle of a store. So, if the store is independent, it is read in cycle 0 of the pipeline from the FPRs and it can go through the alignment stages of the pipeline. If it is dependent on a prior arithmetic operation and it is of the same precision, then it can be either read late in the pipeline or directly fed back to the bottom stage of the pipeline. If it is dependent and of a different precision, then the IDU stalls the store until the prior arithmetic operation finishes, but for most cases, a store executes as though it takes one cycle, whether it is dependent on a prior arithmetic instruction or not.

Another significant performance advantage of the POWER6 processor BFU is in the divide and square root estimate instructions. To help make the divide and square root operations comparable in the number of cycles of execution, an enhanced approximation technique is used. In prior designs, a simple lookup was used, but in the POWER6 processor design, a linear approximation is implemented. This provides better than 14 correct bits for both a reciprocal approximation and a reciprocal square root estimate. A programmer can take immediate advantage of these enhancements if they recompile for the POWER6 processor, as the prior estimate instructions produce only 8 correct bits for the reciprocal and 5 bits for the reciprocal square root. This should save at least one iteration.

In general, the POWER6 processor is an in-order machine, but the BFU instructions can execute slightly out of order. The divide and square root instructions have many empty dispatch slots, and these are utilized by independent BFU instructions. The BFU notifies the IDU when these slots will occur, and the IDU can dispatch in the middle of these slots. If an exception or error occurs in the middle of the execution of the divide, a precise exception can still be achieved by refreshing the machine state from the RU. Thus, there is a mechanism for backing out results from instructions that would never have occurred since the RU only commits machine state in order, though the execution units may update the FPRs and GPRs slightly out of order.

[3] *Unit in the last place* (ulp) is the IEEE term for the least significant bit of the fraction of a floating-point number.

Data fetching

Data fetching is performed by the LSU. The LSU contains two load/store execution pipelines, with each pipeline able to execute a load or store operation in each cycle.

The LSU contains several subunits: the load/store address generation and execution; the L1 D-cache array; and the cache array supporting set-predict and directory arrays, address translation, store queue, load miss queue (LMQ), and data prefetch engine, which perform the following functions:

Load/store execution—In support of the POWER6 processor high-frequency design, the LSU execution pipeline employs a relatively simple dataflow with minimal state machines and hold states. Most load/store instructions are handled by executing a single operation. A hardware state machine is employed to assist in the handling of load/store multiple and string instructions and in the handling of a misaligned load or store. As a result, unaligned data within the 128-byte cache-line boundary is handled without any performance penalty. In the case of data straddling a cache line, the instruction is handled with two internal operations, with the partial data from each stitched together to provide the desired result.

The L1 D-cache load hit pipeline consists of four cycles: approximately one cycle for address generation, two for cache access, and one for load data formatting and transferring to the destination register. In parallel with the L1 D-cache array, address translation is performed by the fully associative, content-addressable-memory-based, 128-entry D-cache ERAT (D-ERAT). Load/stores that miss the D-ERAT initiate a table-walk request to the L2 cache, and LLA mode is entered. The returned data is searched for a matching page table entry, which is then installed in the D-ERAT. The load or store is reexecuted, this time resulting in a D-ERAT hit, and LLA mode is exited. The D-ERAT supports three page sizes concurrently: 4 KB, 64 KB, and 16 MB. The 16-GB page is mapped in the D-ERAT using multiple 16-MB page entries.

Loads that miss the L1 D-cache initiate a cache-line reload request to the L2 cache, and LLA mode is entered. The LMQ tracks the loading of the cache line into the L1 D-cache and supports the forwarding of the load data to the destination register. When the load data returns from L2, the load instruction is reexecuted, the load data is transferred to the destination register, and LLA mode is exited. The LMQ consists of eight entries and is used to track load requests to the L1 D-cache. In SMT mode, two entries would track demand loads (one per thread), and six entries would track some form of prefetch (hardware initiated, software initiated using `dcbt/dcbtst` instructions, or LLA).

The processor core in which the LSU resides runs at twice the frequency of the storage subsystem in which the L2 cache resides. Thus, the store interface from the LSU to the L2 runs at the storage subsystem frequency. The LSU store queue employs store chaining to improve the store bandwidth from the processor core to the storage subsystem. The store queue will chain two successive stores if they are to the same cache line and send them as a single store request to the storage subsystem.

L1 D-cache organization—The POWER6 core contains a dedicated 64-KB, eight-way, set-associative L1 D-cache. The cache-line size is 128 bytes, consisting of four sectors of 32 bytes each. The reload data bus from the L2 cache is 32 bytes. The cache line is validated on a sector basis as each 32-byte sector is returned. Loads can hit against a valid sector before the entire cache line is validated.

The L1 D-cache has two ports that can support either two reads (for two loads) or one write (for a store or cache-line reload). Writes due to cache-line reloads have the highest priority and they block load/store instructions from being dispatched. Reads for executing loads have the next priority. Finally, if there are no cache-line reloads or load reads occurring, completed stores can be written from the store queue to the L1 D-cache. The L1 D-cache is a store-through design: All stores are sent to the L2 cache, and no L1 castouts are required.

The L1 D-cache is indexed with EA bits; the fact that it is 64 KB and eight-way set associative results in 8 KB per set, requiring EA bit 51 to be used to index into the L1 D-cache. With EA bit 51 being above the 4-KB page boundary, EA aliasing conditions exist and have to be handled. To this end, the L1 D-cache directory is indexed with EA(52:56) and organized with visibility to $\mathrm{EA}(51) = \text{'}0\text{'}$ and $\mathrm{EA}(52) = \text{'}1\text{'}$ such that the EA aliasing conditions can be detected.

The L1 D-cache is protected by byte parity; hardware recovery is invoked on detection of a parity error while reading the L1 D-cache. Also, when a persistent hard error is detected either in the L1 D-cache array and its supporting directory or in the set-predict arrays, a set-delete mechanism is used to prohibit the offending set from being validated again. This allows the processor core to continue execution with slightly degraded performance until a maintenance action is performed.

Set predict—To meet the cycle time of the access path, a set-predict array is implemented. The set-predict array is based on the EA and is used as a minidirectory to select which one of the eight L1 D-cache sets contains the load data. Alternatively, the L1 D-cache directory array could be used, but it would take more time, as it is based on the real address (RA) and, thus, would require translation results from the ERAT.

The set-predict array is organized as the L1 D-cache: indexed with EA(51:56) and eight-way set associative. Each entry or set contains 11 EA hash bits, 2 valid bits (one per thread), and a parity bit. The 11-bit EA hash is generated as follows: (EA(32:39) XOR EA(40:47)) plus EA(48:50).

When a load executes, the generated EA(51:56) is used to index into the set-predict array, and EA(32:50) is hashed as described above and compared with the contents of the eight sets of the indexed entry. When an EA hash match occurs and the appropriate thread valid bit is active, the match signal is used as the set select for the L1 D-cache data.

When a cache line is validated, the default mode is a shared mode in which the valid bits are activated for both threads. A nonshared mode is dynamically entered on an entry basis to allow only one thread valid bit to be active. This is beneficial to avoid a thrashing condition in which entries with the same EA hash between threads replace each other in the cache, thus allowing the same EA hash to exist for each thread at the same time.

Accelerator

The POWER6 core implements a vector unit to support the PowerPC VMX instruction set architecture (ISA) and a decimal execution unit to support the decimal ISA. A detailed description of these accelerators is described in a separate paper in this issue [12].

Simultaneous multithreading

The POWER6 processor implements a two-thread SMT for each core. Software thread priority implementation in the POWER6 core is similar to that in the POWER5 core [1]. Additionally, hardware automatically lowers the priority of a thread when it detects that the thread is being stalled. Several features are implemented in the POWER6 core to improve its SMT performance, particularly the following:

- The L1 I-cache and D-cache capacity and associativity as well as the L2 capacity are increased from the POWER5 design.
- The POWER6 core implements an independent dispatch pipe with a dedicated I-buffer and decode logic for each thread. At the dispatch stage, each group of up to five instructions per thread is formed independently and then merged to form a single group of up to seven instructions to be dispatched to the execution units.
- The completion table is dedicated per thread, which allows significantly more outstanding instructions (up to 320 per thread) from both threads.

As a result of these features, the POWER6 core achieves a 15% to 30% speedup improvement in the SMT–to–single thread ratio over the POWER5 design.

Error recovery

The POWER6 core RU contains a copy of the designed state of the processors. Checkpoints of the data in the core are constantly saved in the RU [13]. The data is protected by ECC.

The L2 cache includes the L1 I-cache and D-cache. As a result, the L1 contains temporary data that is protected by parity. Persistent data is in the L2 and L3 caches and is protected by ECC. Major instruction flow, dataflow, and some control logic are checked using parity, residue, or duplication.

When an error is detected, the core is stopped and is prevented from communicating with the rest of the chip. Checkpoint data associated with the last successfully saved instruction from the RU is read and re-stored to the affected core. The L1 I- and D-caches are cleared because the persistent data is protected and stored outside of the core. All translation tables in the core are cleared, as snooping actions are not seen by the core after it has stopped. The processor then resumes at the re-stored checkpoint. If it is determined that the core failure is a hard error (persistent), the content of the RU can be transferred to another idle core in the system, and the task can be restarted on the new core from the checkpoint.

The POWER6 core recovery capability, in conjunction with its extensive error-checking capability, produces an unprecedented level of resiliency for its class of machine [14].

Cache hierarchy

The POWER6 processor cache hierarchy consists of three levels, L1, L2, and L3. The 64-KB L1 I-cache and 64-KB L1 D-cache are integrated in the core, within the IFU and the LSU, respectively. Each core is supported by a private 4-MB L2 cache, which is fully contained on the POWER6 chip. The two cores on a given chip may also share a 32-MB L3 cache. The controller for the L3 cache is integrated in the POWER6 chip, but the data resides off-chip. **Table 6** summarizes the organizations and characteristics of these caches. (The L1 I- and D-caches were described in the section "Processor core.")

L2 cache

The private 4-MB POWER6 processor L2 cache, which comprises 128-byte cache lines and is eight-way set associative, is organized differently than the POWER5 processor L2 cache. Instead of dividing into three independent address-hashed slices, the POWER6 processor L2 cache is operated by a single, centralized controller.

The cache data arrays are organized as four interleaves, with each interleave containing a 32-byte sector of every cache line in the cache. Line fetch operations initiated by

Table 6 POWER6 processor cache attributes.

Cache attribute	*L1 instruction*	*L1 data*	*L2*	*L3*
Capacity	64 KB	64 KB	4 MB	32 MB
Shares (cores)	1	1	1	2
Location	Within core	Within core	On-chip	Off-chip
Line size (bytes)	128	128	128	128
Associativity	4 way	8 way	8 way	16 way
Update policy	Read only	Store through	Store in	Victim
Line inclusion rules	Resides in L2	Resides in L2	None	None
Snooped	No	No	Yes	Yes
Error protection	Parity	Parity	ECC	ECC

the core, castout read operations, and intervention read operations utilize all four interleaves in concert, but read-modify-write operations (performed on behalf of accumulated store-through operations initiated by the core) and writeback operations (which occur when the L2 cache is reloaded) operate only upon one or more 32-byte sectors within a given cache line and, therefore, utilize only the interleaves that are affected by the operations.

Two interleaves, representing the lower 64 bytes in each cache line, are located to the left of the core, while the other two interleaves, representing the upper 64 bytes, are located to the right. Each interleave is composed of 32 SRAM partitions [15]. In aggregate, an interleave can read or write 32 bytes every four cycles of the core. Instead of incurring a varying latency to each of three slices, as in the POWER5 processor L2, the POWER6 processor L2 spreads latency variations across the four interleaves. This results in a constant latency and fixed order for each 32-byte sector returned from the L2 cache to the core.

The directories and coherence management resources are divided into two address-hashed slices. Each directory slice can accept either a core request, a snoop request, or an update operation every other core cycle. Unlike the POWER5 processor L2, which used separate read ports for core and snoop requests, the POWER6 L2 incorporates a sliding window to schedule snoop requests such that they do not collide with core requests, thereby eliminating the need for a second access port in order to reduce power and area.

Both the data arrays and the directory arrays are protected by single-error-correct double-error-detect (SECDED) ECC. In the event of an error, correctable or uncorrectable, soft or hard, the hardware supports autocorrection, auto-invalidation, and autopurge capabilities. With autocorrection, the hardware automatically replaces data or directory information containing a correctable soft error with an error-free copy of the data or directory information. With auto-invalidation, the hardware automatically changes the coherence state of an entry in the cache to invalid if the data or directory information in the entry contains an uncorrectable error or a correctable hard error, and the coherence state of the entry is S, SL, TE, ME, or TEN. With autopurge, the hardware automatically writes a specially marked copy of an entry in the cache back to memory if the data or directory information in the entry contains an uncorrectable error or a correctable hard error, and the coherence state of the entry is T, M, MU, or TN. Both auto-invalidation and autopurge may also mark the directory entry as unusable by setting it to the ID (deleted) state.

The L2 uses an enhanced pseudo-LRU (least recently used) replacement algorithm with biasing toward various invalid locations and away from the most recently accessed I-cache lines.

Because the L1 D-cache is a store-through design in order to reduce accesses to the cache arrays, the L2 accumulates individual core store operations by employing an eight-entry, 128-byte-wide queue per directory slice. All stores gathered by a given entry are presented to the L2 cache with a single read-modify-write operation. This operation uses only the cache interleaves that are affected.

To handle all fetch operations initiated by the core and read-modify-write operations initiated by the L2 store queue, the L2 can employ one of 16 read/claim (RC) machines per directory slice. RC machines manage coherence transactions for all core-initiated cacheable operations. The 32 total machines are needed to enable a sufficient number of outstanding prefetch operations to drive the memory interface to saturation while still handling fetch and store traffic. If a fetch operation hits in the L2 cache, the cache interleaves are read, and data is forwarded to the core. If a read-modify-write operation hits in the L2 cache, the impacted cache interleaves are updated.

If either of these operations misses the L2 cache, the L2 waits for a response from the L3 cache. If the operation hits in the L3, data is returned to the L2 (and possibly independently to the core). If the operation misses the L3, the L2 cache sends a request to the SMP coherence interconnect fabric, and data is eventually returned from another L2, L3, or memory via the SMP data interconnect fabric. For cases in which these operations result in the deallocation of a cache line from the L2 in order to install the newly requested cache line, the L2 must additionally employ one of four castout machines per directory slice to move data or state to the L3 victim cache or to memory. The eight total machines are needed to enable sufficient outstanding castout operations to drive the L3 cache write interface to saturation.

If data is returned from the L3 cache or via the SMP data interconnect fabric, in the case of a fetch, it is forwarded to the core and written to the L2 cache; in the case of a read-modify-write, it is written directly to the nonimpacted L2 cache interleaves and merged with store data prior to writing to the impacted L2 cache interleaves.

To handle incoming snoop requests from the SMP coherence interconnect fabric, the L2 first consults the directory to determine whether it is required to take any action. If so, it can employ one of four snoop machines per directory slice to perform the required task. The eight total machines are needed to enable enough outstanding interventions to drive the datapath that lies between a likely pair of L2 caches to saturation. A snoop machine task might involve reading the data from the cache to send it, via the SMP data interconnect fabric, to another processor and possibly updating the directory state; or reading the data from the cache to send it to memory and invalidating the directory state; or simply updating or invalidating the directory state.

As in the POWER5 processor design, the POWER6 processor L2 cache manages the reservation flags associated with the PowerPC `lwarx`, `ldarx`, `stwcx`, and `stdcx` instructions, and all coherence interactions therewith. Also consistent with the POWER5 processor approach, the design of the POWER6 processor uses a noncacheable unit to manage irregular and complicated storage operations, such as cache-inhibited loads and stores, memory barriers, and translation management operations.

Since the L2 is private and there are two such L2 caches on a chip, it is possible that the data requested by a given core may be found in the L2 cache associated with the other core, leading to an intervention of data between two L2 caches on the same chip. To improve latency in such a scenario, a high-speed cache-to-cache interface was created. Whenever a fetch operation initiated by a core checks its own L2 directory, the operation is forwarded to the other L2 as well. If the operation misses in its own L2, the other L2 directory is checked, and if the line is found, it is read from the other L2 cache and forwarded on a high-speed interface back to the requesting core and L2. This significantly reduces the latency as compared with a normal intervention case.

For some cost-sensitive configurations, single-core POWER6 chips will be used. Such configurations will often have no L3 cache attached. For such cases, a mode of operation is supported in which the unused 4-MB L2 cache associated with a deconfigured core can be configured to accept castouts from the other L2 cache and operate as a victim cache. While this results in a much smaller (4 MB instead of 32 MB) victim cache, it offers significantly better latency, as the victim data is kept on the POWER6 chip.

L3 cache

The 32-MB POWER6 processor L3 cache, which is shared by both cores on a given chip, is 16-way associative and is composed of 128-byte lines, unlike the POWER5 processor L3, which uses 256-byte sectored lines. Also, since both of the private L2 caches are supported by a single L3 victim cache, the rule of exclusivity governing the relationship between the POWER5 processor L2 and L3 caches does not carry over to the design of the POWER6 processor.

The L3 cache data is retained off-chip within one or two specialized embedded DRAM cache chips. Either one or both sets of 8-byte read, 8-byte write, and command buses can be attached to the off-chip storage. These buses can send or receive at a rate of once every other core cycle. When only one L3 data chip is used, the data bandwidth to and from the L3 cache is halved.

For nonsnoop operations, the POWER6 processor L3 is divided into two address-hashed slices. For snoop requests, each directory slice is further divided into two subslices, for a total of four subslices. Each of the two directory slices can accept an L2 fetch-data-read request, an L2 castout data-write request, an update request, or one to two snoop requests every four core cycles (to accept two snoop requests, they must map to different subslices). Like the L2, the POWER6 processor L3 incorporates a sliding window to schedule snoop requests, thereby eliminating any dependence on precise scheduling by the SMP coherence interconnect fabric.

The data arrays on the external cache data chip, the interfaces to and from that chip, and the directory arrays on the POWER6 chip are protected by SECDED ECCs. Like the L2, the L3 supports autocorrection, auto-invalidation, and autopurge capabilities, as well as special invalid states for marking compartments as unusable.

Like the L2, the L3 uses a pseudo-LRU replacement algorithm with biasing toward various invalid cache locations.

To handle all fetch and store operations initiated by the core that miss the L2 and must check the L3 cache, the L3 can employ one of eight read machines per directory slice, private to each L2 cache. The 16 total machines per L2 cache are needed to enable enough outstanding L3 fetch hit operations to drive the L3 cache read interface to saturation. If the data resides in the L3 cache, the read machine retrieves it and routes it back to the requesting L2 cache and core. Otherwise, a response is sent to the requesting L2 cache indicating that it should route the request to the SMP coherent interconnect fabric.

To handle L2 castout write operations, the L3 can employ one of eight write machines per slice, shared by both L2 caches. The number of write machines is tied to the number of castout machines in order to reduce complexity and functional overhead. For cases in which an L2 castout write to the L3 results in the deallocation of a cache line from the L3 and a copy of the line must be moved to memory, the L3 must additionally employ an L3 castout machine associated with the write machine to move data and state to memory. The 16 total machines are needed to enable enough outstanding L3 castout operations to drive the memory write interface to saturation. Since the rule of exclusivity governing the relationship between the POWER5 processor L2 and L3 caches does not carry over to the POWER6 processor design, there are cases in which an L2 castout write must merge state information with a copy of the same cache line in the L3 cache.

To handle incoming snoop requests from the SMP coherence interconnect fabric, the L3 first consults the directory to determine whether it is required to take any action with respect to the snoop operation. If so, it can employ one of four snoop machines per directory subslice to perform the required task. The reason for the number of machines and the possible tasks were described above in the previous section.

Memory subsystem

Each POWER6 chip includes two integrated memory controllers. A memory controller supports up to four parallel channels, each of which can be connected through an off-chip interface to a buffer chip. A channel supports a 2-byte read datapath, a 1-byte write datapath, and a command path that operates four times faster than the DRAM frequency, the fastest of which is 800-MHz DDR2 (double data rate 2) DRAM. Depending on the system configuration, one or both memory controllers may utilize two or four channels each. Each channel may have from one to four buffer chips daisy-chained together. For some system configurations, buffer chips are mounted on the system board, and industry-standard DIMMs (dual inline memory modules) are used. For other configurations, specialized DIMMs directly integrate the buffer chip.

Each buffer chip has a 72-bit bidirectional interface to its attached DIMMs. This 72-bit interface operates at the DRAM frequency. With a *by-4-bit* DRAM configuration, the 72-bit interface is divided into 18 groups of 4 bits. Each group of 4 bits is multidropped to one, two, or four DRAM chips. With a *by-8-bit* DRAM configuration, the 72-bit interface is divided into nine groups of 8 bits. Each group of 8 bits is multidropped to one, two, or four DRAM chips. The maximum frequency degrades because of loading as the number of DRAMs per bus increases.

While the interface between the POWER6 chip and the buffer chips supports 51.2 GB/s of peak read bandwidth and 25.6 GB/s of peak write bandwidth when using 800-MHz DDR2 DRAM with both memory controllers driving four channels each, the sustainable bandwidth is dependent upon the number of daisy-chained buffer chips per channel and the number of DRAM chips multidropped to each buffer chip, as well as the read and write traffic pattern.

Each memory controller is divided into two regions that operate at different frequencies. The first, called the *asynchronous region*, operates at four times the frequency of the attached DRAM, up to a maximum of 3.2 GHz. It manages all interaction with the buffer chips and the DRAM chips attached to them. It tracks and controls the low-level operation flow to maximize utilization while avoiding collisions and it initiates refreshes, controls scrubbing logic, and dynamically powers memory on and memory off.

The second region, called the *synchronous region*, operates at half of the core frequency. It manages all interaction with the SMP coherence and data interconnect fabric, responding as a point of coherence for the memory ranges it maps by servicing reads and writes, arbitrating conflicting requests, and managing directory information for the coherence protocol. The high-level coherent operations that are tracked and managed by the synchronous region of the memory controller are mapped to low-level read and write operations handled by the asynchronous region of the memory controller.

Memory is protected by SECDED ECCs. Scrubbing is employed to find and correct soft, correctable errors. Additionally, IBM Chipkill* technology and redundant bit steering are employed in both by-4-bit and by-8-bit configurations to transparently work around DRAM chip failures. In the POWER6 chip, each channel contains one spare wire for each direction that can be used dynamically to replace a failing bit on a DIMM connector.

I/O subsystem

The POWER6 processor I/O controller is built on the same architectural foundation as that of the POWER4 and POWER5 chips. A 4-byte off-chip read interface and a 4-byte off-chip write interface connect the POWER6 chip to an I/O hub chip. These interfaces operate at one-half of the core frequency but can also run at lower frequency integer divisions of the core frequency to support previous-generation I/O hub chips.

To facilitate concurrent maintenance, the POWER6 processor interrupt presentation function was redesigned to make it decentralized and it was mapped onto the SMP coherence interconnect fabric. A pipelined I/O high-throughput mode was added whereby DMA write operations initiated by the I/O controller are speculatively pipelined. This ensures that in the largest systems, inbound I/O throughput is not limited by the tenure of the coherence phase of the DMA write operations. Partial cache-line DMA write transactions have been redesigned to allow any arbitrary transfer size, from 1 byte to 128 bytes, to complete in a single coherence transaction.

SMP interconnect

While the SMP coherence interconnect fabric for the POWER6 processor design builds upon the nonblocking broadcast transport approach used for the POWER4 and POWER5 processor designs, system considerations resulted in significant topology innovations in addition to support for key coherence protocol innovations.

Topology

As shown in **Figure 6(a)**, the topology of a POWER5 processor-based system consists of a first-level nodal structure composed of up to four POWER5 chips. Coherence links are fully connected such that each chip is directly connected to all of the other chips in the node. Data links form clockwise and counterclockwise rings that connect all of the chips in the node. All of the chips within a node are designed to be packaged within the same multichip module.

Also shown in Figure 6(a), the POWER6 processor first-level nodal structure is composed of up to four POWER6 chips. Relying on the traffic reduction afforded by the innovations in the coherence protocol to reduce packaging overhead, coherence and data traffic share the same physical links by using a time-division-multiplexing (TDM) approach. With this approach, the system can be configured either with 67% of the link bandwidth allocated for data and 33% for coherence or with 50% for data and 50% for coherence. Within a node, the shared links are fully connected such that each chip is directly connected to all of the other chips in the node. There are five 8-byte off-chip SMP interfaces on the POWER6 chip (which operate at half the processor frequency). Three are dedicated for interconnecting the first-level nodal structure, which is composed of up to four POWER6 chips.

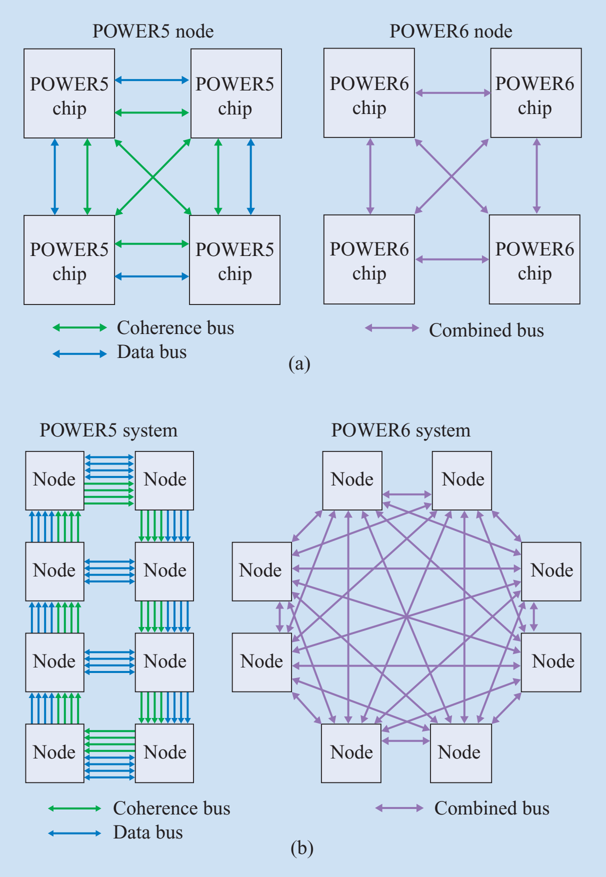

Figure 6

POWER5 and POWER6 processor (a) first-level nodal topology and (b) second-level system topology.

With both the POWER5 and the POWER6 processor approaches, large systems are constructed by aggregating multiple nodes. As shown in **Figure 6(b)**, a POWER5 processor-based system can interconnect up to eight nodes with a parallel ring topology. With this approach, both coherence and data links are organized such that each chip within a node is connected to a corresponding chip in every node by a unidirectional ring. For a system with four chips per node, four parallel rings pass through every node. The POWER5 chip also provides additional data links between nodes in order to reduce the latency and increase the bandwidth for moving data within a system.

While the ring-based topology is ideal for facilitating a nonblocking-broadcast coherence-transport mechanism, it involves every node in the operation of all the other nodes. This makes it more complicated to provide isolation capabilities, which are ideal for dynamic maintenance activities and virtualization.

For POWER6 processor-based systems, as shown in Figure 6(b), the topology was changed to address this issue. Instead of using parallel rings, POWER6 processor-based systems can connect up to eight nodes with a fully connected topology, in which each node is directly connected to every other node. This provides perfect isolation, since any two nodes can interact without involving any other nodes. Also, system latencies do not increase as a system grows from two to eight nodes, yet aggregate system bandwidth increases faster than system size.

Of the five 8-byte off-chip SMP interfaces on the POWER6 chip (which operate at half the processor frequency), the remaining two are dedicated to interconnecting the second-level system structure. Therefore, with a four-chip node, eight such links are available for direct node-to-node connections. Seven of the eight are used to connect a given node to the seven other nodes in an eight-node 64-way system.

The five off-chip SMP interfaces on the POWER6 chip protect both coherence and data with SECDED ECCs.

Coherence transport

The ring-based POWER5 processor-based system topology is ideal for facilitating the nonblocking-broadcast coherence-transport mechanism. Coherence requests travel around the ring attached to the initiating (or master) chip. Each chip attached to the ring broadcasts the request to the other chips in its node. The responses from each snooper in the node are aggregated back at the chip and combined with upstream responses prior to traveling around the same ring, trailing the request. Once all of the responses are gathered and they arrive back at the master chip after completing the circuit, the notification is generated. Like the request, this notification travels around the ring attached to the master chip and is distributed by each chip attached to the ring within its nodes.

With the ring-based POWER5 processor topology, an end-to-end, nonblocking path for request, response, and notification can be established when an initiating chip determines that the attached ring is available. With the fully connected POWER6 processor topology, to achieve similar coherence bandwidth, all of the chips in a node must be coordinated to determine when an end-to-end, nonblocking path can be established for a given request. Furthermore, with the fully connected POWER6 processor-based system topology, there is no circuit to complete, so responses must flow in the direction opposite that of requests and notifications. Therefore, the simple combination of aggregated nodal responses with upstream responses found in the POWER5 processor design is not possible with the POWER6 processor topology.

Beyond the topology-related issues, the enhancements to the coherence protocol required the coherence-transport mechanism to concurrently manage system-wide and scope-limited request, response, and notification broadcasts. The POWER6 processor capability has two supported scopes for scope-limited broadcasts: all chips in the node containing the initiating chip or the initiating chip by itself.

While the topology-related issues, combined with the concurrent multiscope requirements, created conceptual challenges, the team developed innovative approaches to solve them. This enabled the beneficial system characteristics provided by a fully connected nodal topology while maintaining the performance and scalability of the nonblocking-broadcast coherence-transport foundation upon which POWER4 and POWER5 processors were built and extending them, by incorporating the multiscope coherence protocol enhancements.

Datastream prefetching

The POWER6 processor design reduces the effective latency of storage references with a substantial set of design features, including, as described above, three levels of cache, SMT, and LLA. Additionally, it significantly enhances datastream prefetching over its predecessors.

Datastream prefetching in the POWER6 core builds on the stream prefetching capabilities of the previous POWER processors [1, 2, 16] and consists of an enhanced hardware prefetching engine combined with ISA extensions.

The POWER6 core contains 16 stream prefetch request queues (PRQs) compared with eight entries in the POWER5 core, which gives it the capability of tracking and prefetching 16 independent streams. The POWER6 processor also has the capability of detecting and prefetching store streams and load streams. Store streams are analogous to load streams but are defined by a sequence of store instructions rather than load instructions. For load streams, prefetching is split into L1 prefetches and L2 prefetches. All of the prefetches that pass address translation prefetch a line from anywhere in system memory to the appropriate destination: the L1 D-cache for L1 prefetches and the L2 cache for L2 prefetches. The L1 prefetches start prefetching one line ahead of the currently demanded cache line and work up to a steady-state distance of two lines ahead of the line currently being loaded. This two-line prefetching distance is enough to cover the miss latency of the L1 cache. The L2 prefetches are sent ahead of the L1 prefetches with the intention of prefetching from the higher latency levels of the memory hierarchy. To cover the higher latency, L2 prefetches may be sent up to 24 lines ahead of the location of a demand load for a given stream. The requests are sent to read and claim machines in the L2 cache, which process the requests for data and reload the L2 cache for L2 prefetches or return the data back to the core L1 prefetches. In the POWER6 core, each core is supported by 32 read and claim machines, a number sufficient to provide ample bandwidth and to cover the latency to DRAM. The *depth* of the stream, i.e., the number of cache lines ahead of the currently loaded line, may be adjusted in the POWER6 core by changing the default setting of a field in a newly designed special-purpose register called the *datastream control register*, or individually by using the depth field (DEP) of a new version of the `dcbt` or `dcbtst` instruction.

Prefetches terminate on a page boundary (prefetching uses RAs and cannot cross a page boundary because of the RA discontinuities) and, therefore, are most effective with 64-KB and 16-MB pages; these page sizes are also supported in the D-ERAT.

Store-stream prefetching

Store streams are detected and prefetched when the store-stream-enable (SSE) bit in the datastream control register is set. Store prefetching enhances performance for data-intensive applications in which store misses are a factor in the execution of the application. A store miss occurs when a cache line into which an operand is being stored is not in the L2 cache (and is thus also precluded from being in the L1 D-cache). A store miss initiates a read-modify-write sequence on the cache line, which involves fetching the line into the L2 cache and eventually merging the modified data sent from the core into the cache line. Although there are two eight-entry store queues and two four-entry castout queues for each core in the L2, stores that are missing to DRAM may be held up by the latency of the read operation. Store prefetching, like load prefetching, detects the stream and prefetches the lines to mitigate the fetch latency for, in the case of stores, the read-with-intent-to-modify access.

Store prefetching differs from load prefetching in several important aspects. First, store streams are prefetched only into the L2 cache (hence, there are no L1 prefetches for store streams). This is consistent with the nonallocating store-through design of the POWER6 core. Second, store streams are detected by a different procedure than load streams, as an L2-store-miss signal is not available to the data prefetcher. Store streams are allocated only when a multiline window containing the cache-line address of the store does not match any of the 16 PRQs, and the store address is not in the beginning or ending lines of the same multiline window. For each store address, a compare is done against all PRQs, and a matched address advances the state of the stream, just as for load streams. If there is no match, a second compare is done that treats all addresses as if the cache-line size were 2,048 bytes (i.e., an eight-line window compare). If, again, there is no match and the address of the store is not within the first two or last two lines of the eight-line window, a stream entry is created. This method minimizes the creation of duplicate streams, which are prevented for load streams by comparing load addresses with entries in the load-miss queue. Finally, store streams can be subsumed by load streams if a load stream and store stream run into each other. This policy of the higher priority of load streams ensures that L1 prefetches are issued if there are loads in a stream that is also being stored.

The `dcbtst` instruction in the ISA has been enhanced to duplicate the stream-oriented capabilities of the `dcbt` instruction introduced in the POWER5 processor [1]. The `dcbtst` instruction is implemented in the POWER6 processor as a hint to prefetch a cache line or a stream into the L2 cache rather than the L1 cache.

Stream advancement and prefetch profiles

Datastream prefetching is paced by consumption; the loads for a load stream and stores for a store stream are the events that advance a stream to the next state. There are four possible phases for a stream: initial startup, ramp up, steady state, and termination. The initial startup phase includes the detection phase, described in the above section, and the optional initial prefetches, which occur before a stream is confirmed (e.g., one speculative L2 prefetch in the guessed direction). The steady-state phase occurs when prefetching has reached its full depth, as defined by the `DEP` value governing the stream. In the steady-state phase, every line consumed will trigger an L1 prefetch, and every fourth line consumed will trigger two 256-byte (dual-cache-line) L2 prefetches. The L2 prefetches are sent with an indicator that is passed to the memory controller, which optimizes the DIMM access for the 256-byte fetch. The ramp-up phase occurs after the startup phase and is the transitional phase in which prefetches are sent according to a ramp-up profile until the steady-state depth is reached. The highest density of prefetches generated and scheduled occurs in this stage because the prefetch engine seeks to prefetch data the desired number of lines ahead of their use. The duration of this stage depends upon the `DEP` value, the consumption rate of the program, and the length of the stream, which may be determined through the `UNITCNT` field of a `dcbt/dcbtst` instruction or the end of a page. The termination phase also depends upon the stream length. If a stream is either a finite-length stream (as designated by `UNITCNT`) or reaches the end of the page, the L2 prefetches are

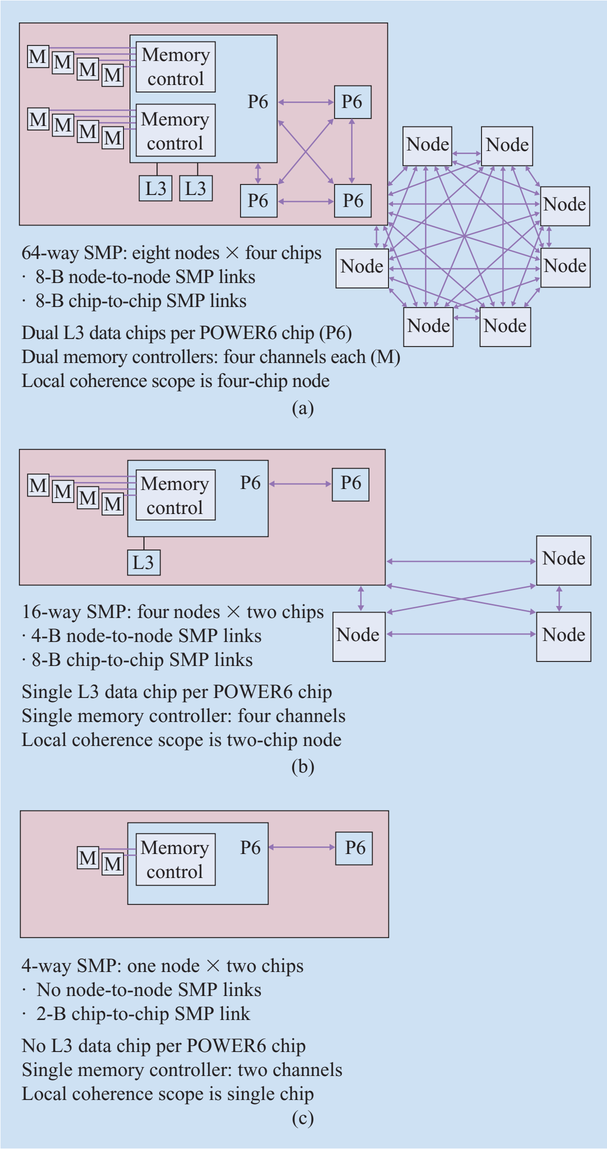
M
M
M
M
Memory control
M
M
M
M
Memory control
P6
L3
L3
P6
P6
P6
Node
Node
Node
Node
Node
Node
Node
Node
64-way SMP: eight nodes × four chips
· 8-B node-to-node SMP links
· 8-B chip-to-chip SMP links
Dual L3 data chips per POWER6 chip (P6)
Dual memory controllers: four channels each (M)
Local coherence scope is four-chip node
(a)
M
M
M
M
Memory control
P6
P6
L3
Node
Node
Node
16-way SMP: four nodes × two chips
· 4-B node-to-node SMP links
· 8-B chip-to-chip SMP links
Single L3 data chip per POWER6 chip
Single memory controller: four channels
Local coherence scope is two-chip node
(b)
M
M
Memory control
P6
P6
4-way SMP: one node × two chips
· No node-to-node SMP links
· 2-B chip-to-chip SMP link
No L3 data chip per POWER6 chip
Single memory controller: two channels
Local coherence scope is single chip
(c)

Figure 7

Examples of how the POWER6 processor balanced system organization and flexibility might be put to use: (a) large, robust system; (b) midrange, robust system; (c) entry-level system.

terminated at that point while the L1 prefetches continue to be sent until they also reach the termination point. A stream may be replaced before it reaches the termination state.

POWER6 balanced system organization

The POWER6 chip offers a wide variety of operational modes to support system configurations ranging from blade servers and entry-level systems to high-end, large-scale servers and consolidation platforms.

Depending upon the compute density and cooling technology of these systems, the clock frequency of POWER6 cores may vary widely. One or both cores on a given chip may be enabled. The L3 cache can be disabled, leaving both sets of off-chip L3 interfaces unused. With the L3 cache enabled, a given POWER6 chip can connect to one or two L3 data chips, depending on bandwidth requirements. These interfaces operate at full width and one-half of the processor frequency.

Of the two memory controllers on the POWER6 chip, none, one, or both may be attached to memory by a set of buffer chips. For each memory controller, two or four channels may be configured. DDR2 memory of varying speeds is supported. Some systems integrate the buffer chip onto the system board and exploit industry-standard DIMMs. Others use specialized DIMMs that integrate the buffer chip.

The SMP fabric interconnect buses may operate at one-half, one-third, or one-fourth of the processor frequency. Of the three buses per POWER6 chip that provide nodal connectivity, zero to three may be connected to provide logical nodes containing one to four fully connected chips. They may operate at a full 8-byte width or at a reduced-pin-count, 2-byte width. System designers may opt for the 2-byte width if the systems are small in SMP size or if they rely heavily upon the multiscope coherence protocol to reduce coherence and data traffic.

Of the two SMP fabric interconnect buses per POWER6 chip that provide system-wide connectivity, zero to two per chip may be connected directly to other nodes. They may operate at full 8-byte width or at a reduced-pin-count, 4-byte width. System designers may opt for the 4-byte width if the systems are moderate in SMP size or if they rely to some degree upon the multiscope coherence protocol.

We have described the flexibility of the POWER6 processor balanced system organization. We next present some examples of how that flexibility might be used. **Figure 7(a)** illustrates the potential for a large, robust, 64-way system that uses 8-byte SMP interconnect links, both L3 data ports to maximize L3 bandwidth, and all eight memory channels per chip. **Figure 7(b)** highlights a midrange, robust, 16-way system that uses 8-byte and 4-byte SMP interconnect links, a single L3 data port, and four memory channels. **Figure 7(c)** illustrates a four-way, low-cost, entry-level system that connects two chips using a 2-byte SMP interconnect link, has no L3 cache, and supports two memory channels. **Table 7** compares the signal I/O counts used for each POWER6 chip in each of these systems.

Summary

With its high-frequency core architecture, enhanced SMT capabilities, balanced system throughput, and scalability

Table 7 POWER6 processor functional signal I/O comparison for various systems.

Function I/O group	*Type of system*		
	Large robust	*Midrange robust*	*Entry level*
Memory interfaces (I/Os)	~400	~200	~100
SMP fabric interfaces (I/Os)	~900	~400	~70
L3 cache interfaces (I/Os)	~420	~210	0
I/O subsystem interfaces (I/Os)	~100	~100	~100
Total functional I/Os	~1,820	~910	~270

extensions, the POWER6 microprocessor provides higher levels of performance than the predecessor POWER5 microprocessor-based systems while offering greater flexibility in system packaging trade-offs. Additionally, improvements in functionality, RAS, and power management have resulted in valuable new characteristics of POWER6 processor-based systems.

Acknowledgments

The authors acknowledge the teams from several IBM research and development laboratories around the world that designed, developed, verified, and tested the POWER6 chipset and systems. It is the innovation and dedication of these people that has transformed the POWER6 microprocessor from vision to reality.

*Trademark, service mark, or registered trademark of International Business Machines Corporation in the United States, other countries, or both.

**Trademark, service mark, or registered trademark of Sun Microsystems, Inc., in the United States, other countries, or both.

References

1. B. Sinharoy, R. N. Kalla, J. M. Tendler, R. J. Eickemeyer, and J. B. Joyner, "POWER5 System Microarchitecture," *IBM J. Res. & Dev.* **49**, No. 4/5, 505–521 (2005).
2. J. M. Tendler, J. S. Dodson, J. S. Fields, Jr., H. Le, and B. Sinharoy, "POWER4 System Microarchitecture," *IBM J. Res. & Dev.* **46**, No. 1, 5–25 (2002).
3. W. J. Armstrong, R. L. Arndt, T. R. Marchini, N. Nayar, and W. M. Sauer, "IBM POWER6 Partition Mobility: Moving Virtual Servers Seamlessly Between Physical Systems," *IBM J. Res. & Dev.* **51**, No. 6, 757–762 (2007, this issue).
4. T. N. Buti, R. G. McDonald, Z. Khwaja, A. Ambekar, H. Q. Le, W. E. Burky, and B. Williams, "Organization and Implementation of the Register-renaming Mapper for Out-of-order IBM POWER4 Processors," *IBM J. Res. & Dev.* **49**, No. 1, 167–188 (2005).
5. D. Lenoski, J. Laudon, K. Gharachorloo, W.-D. Weber, A. Gupta, J. Hennessy, M. Horowitz, and M. S. Lam, "The Stanford Dash Multiprocessor," *Computer* **25**, No. 3, 63–79 (1992).
6. B. Curran, B. McCredie, L. Sigal, E. Schwarz, B. Fleischer, Y.-H. Chan, D. Webber, M. Vaden, and A. Goyal, "4GHz+ Low-Latency Fixed-Point and Binary Floating-Point Execution Units for the POWER6 Processor," *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC)*, San Francisco, CA, February 2006, pp. 1712–1734.
7. R. Kalla, B. Sinharoy, and J. M. Tendler, "IBM POWER5 Chip: A Dual-Core Multithreaded Processor," *IEEE Micro* **24**, 40–47 (2004).
8. E. M. Schwarz, M. Schmookler, and S. D. Trong, "FPU Implementations with Denormalized Numbers," *IEEE Transactions on Computers* **54**, No. 7, 825–836 (2005).
9. X. Y. Yu, Y.-H. Chan, M. Kelly, E. Schwarz, B. Curran, and B. Fleischer, "A 5GHz+ 128-bit Binary Floating-Point Adder for the POWER6 Processor," *Proceedings of the 32nd European Solid-State Circuits Conference*, Montreux, Switzerland, September 2006; see *http://www.ece.ucdavis.edu/~yanzi/esscirc06_submit.pdf*.
10. S. D. Trong, M. Schmookler, E. M. Schwarz, and M. Kroener, "P6 Binary Floating-Point Unit," *Proceedings of the 18th IEEE Symposium on Computer Arithmetic*, Montpellier, France, 2007, pp. 77–86.
11. E. M. Schwarz, "Binary Floating-Point Unit Design: The Fused Multiply-add Dataflow," *High-Performance Energy-Efficient Microprocessor Design*, V. G. Oklobdzija and R. K. Krishnamurthy, Eds., Springer, Dordrecht, The Netherlands, 2006, pp. 189–208.
12. L. Eisen, J. W. Ward III, H.-W. Tast, N. Mäding, J. Leenstra, S. M. Mueller, C. Jacobi, J. Preiss, E. M. Schwarz, and S. R. Carlough, "IBM POWER6 Accelerators: VMX and DFU," *IBM J. Res. & Dev.* **51**, No. 6, 663–683 (2007, this issue).
13. M. J. Mack, W. M. Sauer, S. B. Swaney, and B. G. Mealey, "IBM POWER6 Reliability," *IBM J. Res. & Dev.* **51**, No. 6, 763–774 (2007, this issue).
14. J. W. Kellington, R. McBeth, P. Sanda, and R. N. Kalla, "IBM® POWER6™ Processor Soft Error Tolerance Analysis Using Proton Irradiation," *Proceedings of the IEEE Workshop on Silicon Errors in Logic—Systems Effects (SELSE) Conference*, Austin, TX, April 2007; see *http://www.selse.org/Papers/28_Kellington_P.pdf*.
15. D. W. Plass and Y. H. Chan, "IBM POWER6 SRAM Arrays," *IBM J. Res. & Dev.* **51**, No. 6, 747–756 (2007, this issue).
16. F. P. O'Connell and S. W. White, "POWER3: The Next Generation of PowerPC Processors," *IBM J. Res. & Dev.* **44**, No. 6, 873–884 (2000).

Received January 12, 2007; accepted for publication March 9, 2007; Internet publication October 23, 2007

Hung Q. Le *IBM Systems and Technology Group, 11400 Burnet Road, Austin, Texas 78758 (hung@us.ibm.com)*. Mr. Le is a Distinguished Engineer in the POWER* microarchitecture development team of the Systems and Technology Group. He joined IBM in 1979 after graduating from Clarkson University with a B.S. degree in electrical and computer engineering. He has worked on the development of several IBM mainframe and POWER and PowerPC processors. His technical interests are in the field of processor design involving multithreading, superscalar, and out-of-order design.

William J. Starke *IBM Systems and Technology Group, 11400 Burnet Road, Austin, Texas 78758 (wstarke@us.ibm.com)*. Mr. Starke is a Senior Technical Staff Member in the POWER development team of the Systems and Technology Group. He joined IBM in 1990 after graduating from Michigan Technological University with a B.S. degree in computer science. After several years of cache hierarchy and symmetric multiprocessor (SMP) hardware performance analysis for both IBM mainframe and POWER server development programs, he transitioned to logic design and microarchitecture development, working initially on the POWER4 and POWER5 programs. Mr. Starke led the development of the POWER6 cache hierarchy and SMP interconnect, and now serves as the Chief Architect for the POWER7* storage hierarchy.

J. Stephen Fields *IBM Systems and Technology Group, 11400 Burnet Road, Austin, Texas 78758 (sfields@us.ibm.com)*. Mr. Fields is a Distinguished Engineer in the POWER development team of the Systems and Technology Group. He joined IBM in 1988 after graduating from the University of Illinois with a B.S. degree in electrical engineering. He has worked on a variety of development efforts ranging from the IBM Micro Channel*, Peripheral Component Interface, and memory controllers, and he has been working on microprocessor cache hierarchy and SMP development since the POWER4 program. Mr. Fields currently is responsible for post-silicon validation for POWER6 and POWER7 processors.

Francis P. O'Connell *IBM Systems and Technology Group, 11400 Burnet Road, Austin, Texas 78758 (oconnell@us.ibm.com)*. Mr. O'Connell is a Senior Technical Staff Member in the POWER system development area. For the past 22 years, he has focused on scientific and technical computing performance within IBM, including microprocessor and systems design, compiler performance, algorithm development, and application tuning. Mr. O'Connell joined IBM in 1981 after receiving a B.S. degree in mechanical engineering from the University of Connecticut. He subsequently earned an M.S. degree in engineering-economic systems from Stanford University.

Dung Q. Nguyen *IBM Systems and Technology Group, 11400 Burnet Road, Austin, Texas 78758*. Mr. Nguyen is a Senior Engineer in the POWER development team of the Systems and Technology Group. He joined IBM in 1986 after graduating from the University of Michigan with an M.S. degree in materials engineering. He has worked on the development of several processors including POWER3*, POWER4, POWER5, and POWER6. He is currently working on the POWER7 microprocessor. Mr. Nguyen's technical interests are in the field of processor design involving instruction sequencing and multithreading.

Bruce J. Ronchetti *IBM Systems and Technology Group, 11400 Burnet Road, Austin, Texas 78758 (ronchett@us.ibm.com)*. Mr. Ronchetti is a Senior Technical Staff Member in the POWER system development area. For the past ten years, he has focused on processor core microarchitecture development, particularly in load and store units. Mr. Ronchetti joined IBM in 1979 after receiving a B.S. degree in electrical engineering from Lafayette College.

Wolfram M. Sauer *IBM Systems and Technology Group, 11400 Burnet Road, Austin, Texas 78758 (wsauer@us.ibm.com)*. Mr. Sauer is a Senior Technical Staff Member in the processor development area. He received a diploma degree (Diplom-Informatiker) in computer science from the University of Dortmund, Germany, in 1984. He subsequently joined IBM at the development laboratory in Boeblingen, Germany, and worked on the S/370* (later S/390* and zSeries*) processor design, microcode, and tools. He joined IBM Austin in 2002 to work on the POWER6 project.

Eric M. Schwarz *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (eschwarz@us.ibm.com)*. Dr. Schwarz is a Distinguished Engineer in zSeries, iSeries*, and pSeries* processor development. He received a B.S. degree in engineering science from The Pennsylvania State University, an M.S. degree in electrical engineering from Ohio University, and a Ph.D. degree in electrical engineering from Stanford University. He joined IBM at the Endicott Glendale Laboratories working on follow-ons to the Enterprise System/4381* and Enterprise System/9370* computers. He later worked on the G4, G5, G6, z900, z990, z9* 109, and POWER6 processor-based computers. He led the development of floating-point units for all these computers and was also Chief Engineer of the z900. Dr. Schwarz is active in the IEEE Symposium on Computer Arithmetic and has been on the program committee since 1993.

Michael T. (Mike) Vaden *IBM Systems and Technology Group, 11400 Burnet Road, Austin, Texas 78758 (mtvaden@us.ibm.com)*. Mr. Vaden is a Senior Engineer. He has worked on many of the POWER and PowerPC processors including the logic design for the fixed-point unit in the RIOS Single Chip, PowerPC 601 microprocessor, POWER5 and POWER6 processors, and the L2 cache control logic for the POWER3 and POWER3+ processors. Mr. Vaden holds a B.S.E.E degree from Texas A&M University and an M.S.E.E degree from the University of Texas at Austin.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.