POWER4 system microarchitecture

by J. M. Tendler
J. S. Dodson
J. S. Fields, Jr.
H. Le
B. Sinharoy

The IBM POWER4 is a new microprocessor organized in a system structure that includes new technology to form systems. The name *POWER4* as used in this context refers not only to a chip, but also to the structure used to interconnect chips to form systems. In this paper we describe the processor microarchitecture as well as the interconnection architecture employed to form systems up to a 32-way symmetric multiprocessor.

Introduction

IBM announced the RISC System/6000* (RS/6000*, the predecessor of today's IBM eServer pSeries*) family of processors in 1990. The initial models ranged in frequency from 20 MHz to 30 MHz [1] and achieved performance levels exceeding those of many of their contemporaries [2]. In 1993, the POWER2 microprocessor was introduced, operating initially at frequencies ranging from 55 MHz to 71.5 MHz [3]. The POWER2 microprocessor was capable of simultaneous execution of up to six instructions per clock cycle. The PowerPC 601* microprocessor was also introduced in 1993 [4]. The PowerPC 601 resulted from a joint development effort by IBM, Apple, and Motorola at the Somerset Design Center in Austin, Texas. Like their predecessors, systems built using these microprocessors were 32-bit architectures, though the PowerPC Architecture* did recognize a 64-bit implementation. With the RS64 microprocessor introduced in 1997 and the POWER3 microprocessor introduced in 1998 [5], 64-bit architected UNIX**-based RS/6000 systems became commonplace. The RS64 and its follow-on microprocessors, the RS64-II [6], RS64-III [7], and RS64-IV [8], were optimized for commercial applications. The RS64 initially appeared in systems operating at 125 MHz. Most recently, the RS64-IV has been shipping in systems operating at up to 750 MHz. The POWER3 and its follow-on, the POWER3-II [9], were optimized for technical applications. Initially introduced at 200 MHz, most recent systems using the POWER3-II have been operating at 450 MHz. The RS64 microprocessor and its follow-ons were also used in AS/400* systems (the predecessor to today's eServer iSeries*).

POWER4 was designed to address both commercial and technical requirements. It implements and extends in a compatible manner the 64-bit PowerPC Architecture [10]. First used in pSeries systems, it will be staged into the iSeries at a later date. It leverages IBM technology using an 0.18-μm-lithography copper and silicon-on-insulator (SOI) technology [11]. In the ongoing debate between the "speed demons" (high clock rate) and the "braniacs" (more complex design but a higher instructions-per-cycle rate [12]), IBM UNIX-based systems have traditionally been in the braniac camp. With the POWER4, IBM opted to also embrace the speed-demon approach. The POWER4 design was initially introduced at processor frequencies of 1.1 GHz and 1.3 GHz, surpassing all other 64-bit microprocessors in some key performance benchmarks [13].

In this paper we first describe the objectives that were established for the design. We then take a closer look at the components of the resultant systems from a

0018-8646/02/$5.00 © 2002 IBM

microarchitecture perspective. We begin by first discussing the POWER4 chip. One of IBM's major strengths is its system expertise—the ability to design multiple parts in a consistent and synergistic manner. In that light, POWER4 cannot be considered only a chip, but rather an architecture within which a set of chips are designed together to realize a system. As such, POWER4 can be considered a technology in its own right. In that light, we discuss how systems are built by interconnecting POWER4 chips to form symmetric multiprocessors (SMPs) of any size up to a 32-way SMP. The interconnect topology, referred to as a distributed switch, is new to the industry. Finally, no system discussion would be complete without some view of the reliability, availability, and serviceability (RAS) features and philosophy incorporated into POWER4 systems. A more complete discussion of the POWER4 RAS features may be found in a companion paper by Bossen et al. [14].

Guiding principles

In designing POWER4, the engineering team was guided by a set of principles which included the following:

- *SMP optimization:* The server must be designed for high-throughput multitasking environments. The system must be optimized for SMP operation, in which a large number of transistors can be used to improve total system performance. By their very nature, servers process disparate information, often with multiple processes active concurrently.
- *Full-system design approach:* To optimize the system, we began with the full design in mind up front. This marriage of process technology, packaging, and microarchitecture was designed to allow software to exploit them. The processor was designed to fit effectively in this environment. We designed the entire system together, from the processor to memory and I/O bridge chips. A new high-performance processor needs a new subsystem to feed it effectively.
- *Very-high-frequency design:* To maintain a leadership position, we planned from the outset to deliver best-of-breed operating frequencies. We revamped our chip design with new transistor-level tools [15], and have transformed complex control logic into regular dataflow constructs. Additionally, we have designed the system to permit system balance to be preserved as technology improvements become available, allowing even higher processor frequencies to be delivered.
- *Leadership in reliability, availability, and serviceability (RAS):* Servers have evolved toward continuous operation. We had already begun designing into our systems RAS attributes previously seen only in mainframe systems. With POWER4, we accepted the principle that one does not achieve high levels of reliability only by choosing reliable parts and incorporating error-correction-code (ECC) logic into major arrays. Instead, we required an approach that eliminated outages and provided redundancy in cases where errors could not be eliminated. Where possible, if an error occurred, we worked to transform hard machine stops (checkstops) into synchronous machine interrupts to software to allow the system to circumvent problems if possible.
- *Balanced technical and commercial performance:* As we balanced the system, we made sure the design could handle a varied and robust set of workloads. This is especially important as the e-business world evolves and data-intensive demands on systems merge with commercial requirements. The need to satisfy high-performance computing requirements with their historical high-bandwidth demands and commercial requirements with their data-sharing and SMP scaling requirements dictated a single design to address both environments. This would also allow us to meet the needs of what became pSeries and iSeries eServers with a single design.
- *Binary compatibility:* Several internal IBM task forces in the first half of the 1990s had concluded that the PowerPC Architecture had no technical impediments to prevent it from scaling up to significantly higher frequencies with excellent performance. With no technical reason to change, in order to keep our customers' software investment intact, we accepted the absolute requirement of maintaining binary compatibility for both 32-bit and 64-bit applications with prior PowerPC* and PowerPC AS systems.

POWER4 chip

The components of the POWER4 chip are shown in **Figure 1**. The design features two processors on one chip; included in what we are referring to as the processor are the various execution units and the split first-level instruction and data caches. The two processors share a unified second-level cache, also on the same chip, through a core interface unit (CIU), as shown in Figure 1. The CIU is a crossbar switch between the L2, implemented as three separate, autonomous cache controllers, and the two processors. Each L2 cache controller can operate concurrently and feed 32 bytes of data per cycle. The CIU connects each of the three L2 controllers to either the data cache or the instruction cache in either of the two processors. Additionally, the CIU accepts stores from the processors across 8-byte-wide buses and sequences them to the L2 controllers. Each processor has associated with it a noncacheable unit, the NC unit in Figure 1, responsible for handling instruction-serializing functions and performing any noncacheable operations in the storage hierarchy. Logically, this is part of the L2.

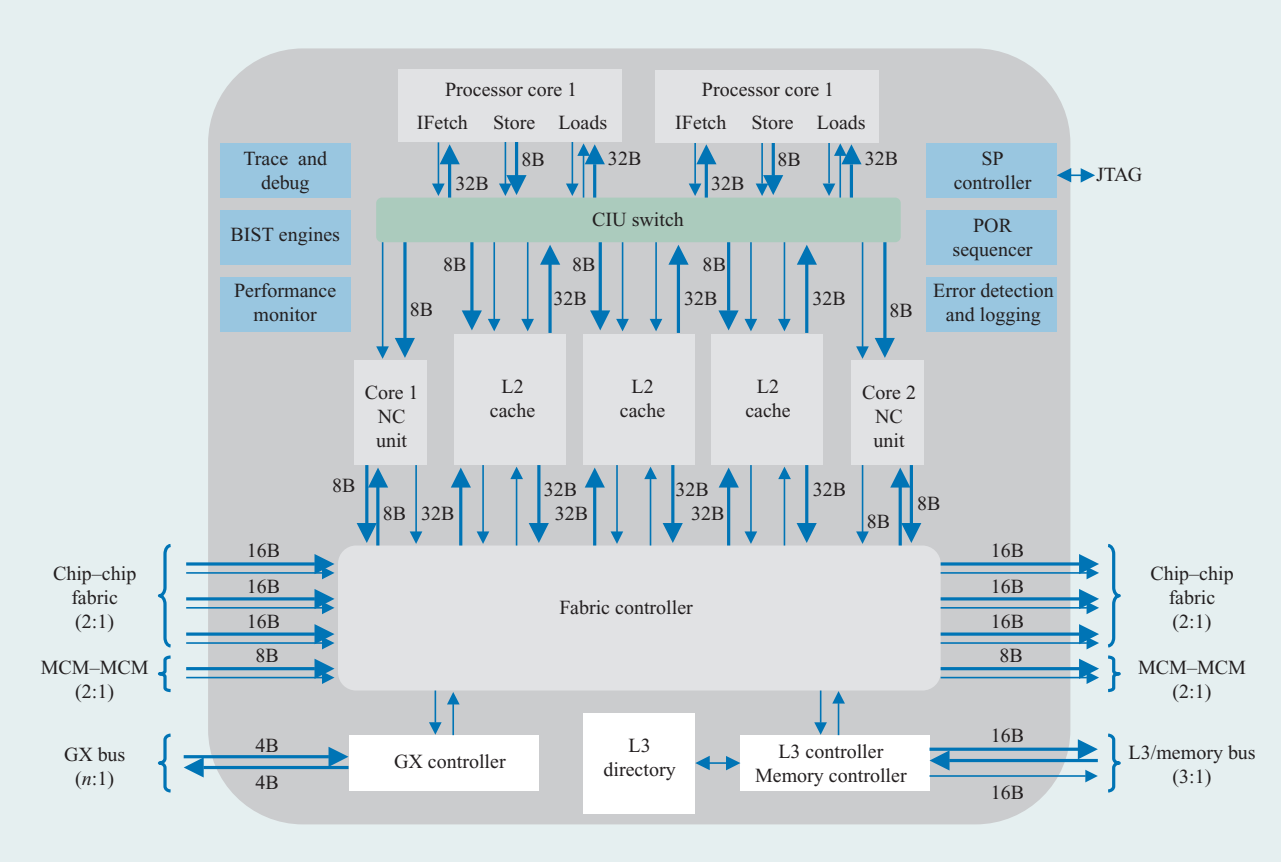

Figure 1

POWER4 chip logical view.

The directory for a third-level cache, L3, and its controller are also located on the POWER4 chip. The actual L3 is on a separate chip. A separate functional unit, referred to as the fabric controller, is responsible for controlling dataflow between the L2 and L3 controller for the chip and for POWER4 communication. The GX controller is responsible for controlling the flow of information into and out of the system. Typically, this would be the interface to an I/O drawer attached to the system. With the POWER4 architecture, however, this is also the point at which we would directly attach an interface to a switch for clustering multiple POWER4 nodes.

Also included on the chip are functions we logically call pervasive function. These include trace and debug facilities used for first-failure data capture, built-in self-test (BIST) facilities, a performance-monitoring unit, an interface to the service processor (SP) used to control the overall system, power-on reset (POR) sequencing logic, and error detection and logging circuitry.

A die photo of the POWER4 chip is shown in **Figure 2**. The chip contains 174 million transistors interconnected across seven layers of copper metallurgy. For clarity, the major components of the chip are labeled. Their operation is described in the remainder of this paper.

Four POWER4 chips can be packaged on a single module to form an eight-way SMP. Four such modules can be interconnected to form a 32-way SMP. To accomplish this, each chip contains five primary interfaces. To communicate with other POWER4 chips on the same module, there are logically four 16-byte buses. Physically, these four logical buses are implemented with six buses, three on and three off, as shown in Figure 1. To communicate with POWER4 chips on other modules, there are two 8-byte buses, one on and one off. Each chip has its own interface to the off-chip L3 across two 16-byte-wide buses, one on and one off, operating at one-third processor frequency. To communicate with I/O devices and other computing nodes, two 4-byte-wide GX buses, one on and one off, operating at one-third processor

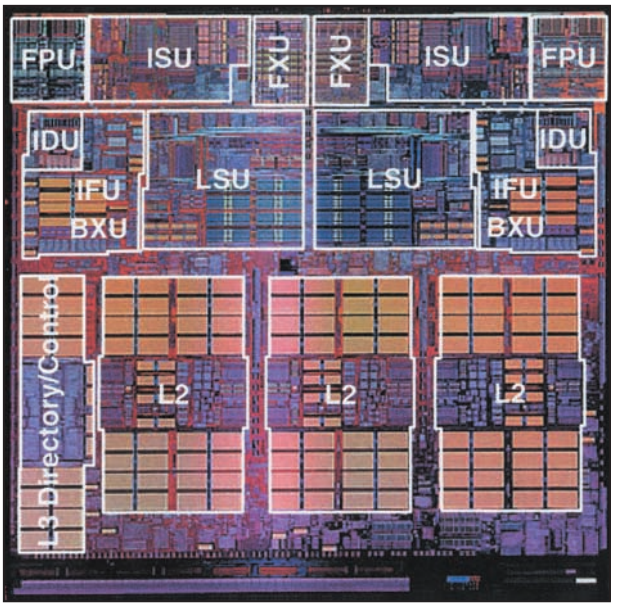

Figure 2

POWER4 chip photograph showing the principal functional units in the microprocessor core and in the memory subsystem.

frequency, are used. Finally, each chip has its own JTAG interface to the system service processor. All of the above buses, except for the JTAG interface, scale with processor frequency. POWER4 systems will be offered at more than one frequency. It is also anticipated that technological advances will allow us to increase processor frequency over time. As this occurs, bus frequencies will scale proportionately, allowing system balance to be maintained.

POWER4 processor

Figure 3 shows a high-level block diagram of a POWER4 processor. The two processors on the chip are identical and provide a two-way SMP model to software. The internal microarchitecture of the core is a speculative superscalar out-of-order execution design. Up to eight instructions can be issued each cycle, with a sustained completion rate of five instructions. Register-renaming pools and other out-of-order resources coupled with the pipeline structure allow the design to have more than 200 instructions in flight at any given time. In order to exploit instruction-level parallelism, there are eight execution units, each capable of being issued an instruction each cycle. Two identical floating-point execution units, each capable of starting a fused multiply and add each cycle [i.e., a maximum four floating-point operations (FLOPs) per cycle per processor] are provided. In order to feed the dual floating-point units, two load/store units, each capable of performing address-generation arithmetic, are provided. Additionally, there are dual fixed-point execution units, a branch execution unit, and an execution unit to perform logical operations on the condition register.

Branch prediction

To help mitigate the effects of the long pipeline necessitated by the high-frequency design, POWER4 invests heavily in branch-prediction mechanisms. Branch-prediction schemes are not new [16, 17]. POWER4 uses a multilevel branch-prediction scheme to predict whether or not a conditional branch instruction is taken. Similar schemes have been implemented in other systems; see [18] for example. Additionally, branch target addresses are predicted for those instructions that branch to an address specified in either the count register or the link register. Similar concepts have been discussed in the literature [19, 20].

In POWER4, up to eight instructions are fetched each cycle from the direct-mapped 64KB instruction cache. The branch-prediction logic scans the fetched instructions, looking for up to two branches each cycle. Depending upon the branch type found, various branch-prediction mechanisms engage to help predict the branch direction or the target address of the branch or both. Branch direction for unconditional branches is not predicted. All conditional branches are predicted, even if the condition register bits upon which they are dependent are known at instruction fetch time. Branch target addresses for the PowerPC branch-to-link-register (bclr) and branch-to-count-register (bcctr) instructions are predicted using a hardware-implemented link stack and count cache mechanism, respectively. Target addresses for absolute and relative branches are computed directly as part of the branch scan function.

As branch instructions flow through the rest of the pipeline and ultimately execute in the branch-execution unit, the actual outcomes of the branches are determined. At that point, if the predictions were found to be correct, the branch instructions are simply completed like all other instructions. If a prediction is found to be incorrect, the instruction-fetch logic causes the mispredicted instructions to be discarded and begins refetching instructions along the corrected path.

POWER4 uses a set of three branch-history tables to predict the direction of branch instructions. The first table, called the local predictor, is similar to a traditional branch-history table (BHT). It is a 16 384-entry array indexed by the branch instruction address producing a 1-bit predictor that indicates whether the branch direction should be *taken* or *not taken*. The second table, called the global predictor, predicts the branch direction on the basis of the actual path of execution to reach the branch. The path of execution is identified by an 11-bit vector, one bit per group of instructions fetched from the instruction

cache for each of the previous eleven fetch groups. This vector is referred to as the global history vector. Each bit in the global history vector indicates whether or not the next group of instructions fetched are from a sequential cache sector. The global history vector captures this information for the actual path of execution through these sectors. That is, if there is a redirection of instruction fetching, some of the fetched group of instructions are discarded and the global history vector is immediately corrected. The global history vector is hashed, using a bitwise exclusive or with the address of the branch instruction. The result indexes into a 16 384-entry global history table to produce another 1-bit branch-direction predictor. Like the local predictor, this 1-bit global predictor indicates whether the branch should be predicted to be taken or not taken. Finally, a third table, called the selector table, keeps track of which of the two prediction schemes works better for a given branch and is used to select between the local and the global predictions. The 16 384-entry selector table is indexed exactly the same way as the global history table to produce the 1-bit selector. This combination of branch-prediction tables has been shown to produce very accurate predictions across a wide range of workload types.

If the first branch encountered in a particular cycle is predicted as not taken and a second branch is found in the same cycle, POWER4 predicts and acts on the second branch in the same cycle. In this case, the machine registers both branches as predicted, for subsequent resolution at branch execution, and redirects the instruction fetching on the basis of the second branch.

As branch instructions are executed and resolved, the branch-history tables and the other predictors are updated to reflect the latest and most accurate information. Dynamic branch prediction can be overridden by software. This is useful for cases in which software can predict branches better than the hardware. It is accomplished by setting two previously reserved bits in conditional branch instructions, one to indicate a software override and the other to predict the direction. When these two bits are zero (suggested use for reserved bits), hardware branch prediction is used. Since only reserved bits are used for this purpose, 100% binary compatibility with earlier software is maintained.

POWER4 uses a link stack to predict the target address for a branch-to-link instruction that it believes corresponds to a subroutine return. By setting the hint bits in a branch-to-link instruction, software communicates to the processor whether a branch-to-link instruction represents a subroutine return, a target address that is likely to repeat, or neither. When instruction-fetch logic fetches a branch-and-link instruction (either conditional or unconditional) predicted as taken, it pushes the address of the next instruction onto the link stack. When it fetches a

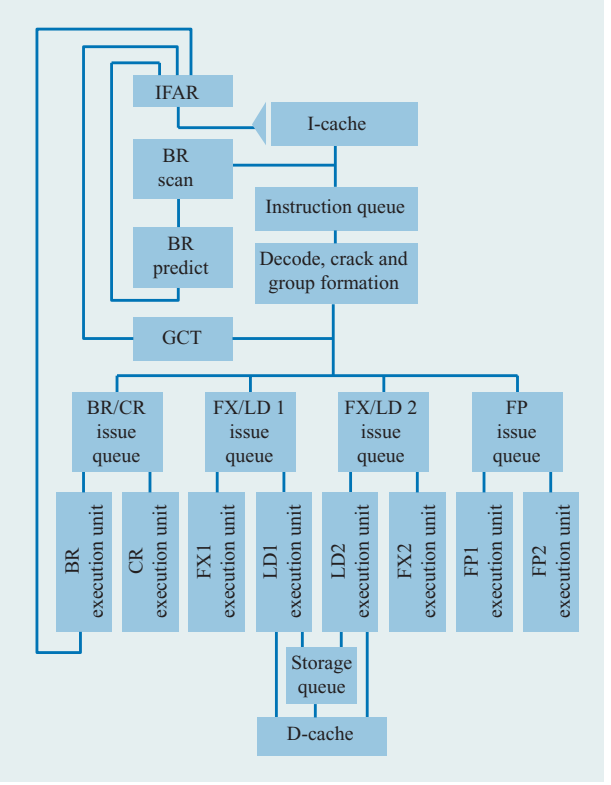

Figure 3

POWER4 core.

branch-to-link instruction with taken prediction and with hint bits indicating a subroutine return, the link stack is popped, and instruction fetching starts from the popped address.

To preserve the integrity of the link stack in the face of mispredicted branch target link instructions, POWER4 employs an extensive speculation tolerance mechanism in its link stack implementation to allow recovery of the link stack under most circumstances.

The target address of a branch-to-count instruction is often repetitive. This is also true for some of the branch-to-link instructions that are not predictable through the use of the link stack (because they do not correspond to a subroutine return). By setting the hint bits appropriately, software communicates to the hardware whether the target addresses for such branches are repetitive. In these cases, POWER4 uses a 32-entry, tagless, direct-mapped cache, called a count cache, to predict the repetitive targets, as indicated by the software hints. Each entry in the count cache can hold a 62-bit address. When a branch-to-link or branch-to-count instruction is executed, for which the software indicates that the target is repetitive and therefore predictable, the target address is written in the

count cache. When such an instruction is fetched, the target address is predicted using the count cache.

Instruction fetch

Instructions are fetched from the instruction cache (I-cache) on the basis of the contents of the instruction-fetch address register (IFAR). The IFAR is normally loaded with an address determined by the branch-prediction logic. As noted earlier, for cases in which the branch-prediction logic is in error, the branch-execution unit will cause the IFAR to be loaded with the corrected address of the instruction stream to be fetched. Additionally, there are other factors that can cause a redirection of the instruction stream, some based on internal events, others on interrupts from external events. In any case, once the IFAR is loaded, the I-cache is accessed and retrieves up to eight instructions per cycle. Each line in the I-cache can hold 32 PowerPC instructions, i.e., 128 bytes. Each line is divided into four equal sectors. Since I-cache misses are infrequent, to save area, the I-cache has been designed to contain a single port that can be used to read or write one sector per cycle. The I-cache directory (IDIR) is indexed by the effective address and contains 42 bits of real address per entry.

On an I-cache miss, instructions are returned from the L2 in four 32-byte transmissions. The L2 normally returns the critical sector (the sector containing the specific word address that references the cache line) in one of the first two beats. Instruction-fetch logic forwards these demand-oriented instructions into the pipeline as quickly as possible. In addition, the cache line is written into one entry of the instruction-prefetch buffer so that the I-cache itself is available for successive instruction fetches. The instructions are written to the I-cache during cycles when instruction fetching is not using the I-cache, as is the case when another I-cache miss occurs. In this way, writes to the I-cache are hidden and do not interfere with normal instruction fetching.

The PowerPC Architecture specifies a translation-lookaside buffer (TLB) and a segment-lookaside buffer (SLB) to translate from the effective address (EA) used by software and the real address (RA) used by hardware to locate instructions and data in storage. Since these translation mechanisms take several cycles, once translated, the EA, RA pair is stored in a 128-entry, two-way set-associative array, called the effective-to-real address translation (ERAT) table. POWER4 implements separate ERATs for instruction-cache (IERAT) and data-cache (DERAT) accesses. Both ERATs are indexed using the effective address. A common 1024-entry four-way set-associative TLB is implemented for each processor.

When the instruction pipeline is ready to accept instructions, the IFAR content is sent to the I-cache, IDIR, IERAT, and branch-prediction logic. The IFAR is updated with the address of the first instruction in the next sequential sector. In the next cycle, instructions are received from the I-cache and forwarded to an instruction queue from which the decode, crack, and group formation logic shown in Figure 3 pulls instructions. This is done even before it is known that there is a I-cache hit. Also received in the same cycle are the RA from the IDIR, the EA, RA pair from the IERAT, and the branch-direction-prediction information. The IERAT is checked to ensure that it has a valid entry and that its RA matches the contents of the IDIR. If the IERAT has an invalid entry, the EA must be translated from the TLB and SLB. Instruction fetching is then stalled. Assuming that the IERAT is valid, if the RA from the IERAT matches the contents of the IDIR, an I-cache hit is validated. Using the branch-prediction logic, the IFAR is reloaded and the process is repeated. Filling the instruction queue in front of the decode, crack, and group formation logic allows instruction fetching to run ahead of the rest of the system and queue work for the remainder of the system. In this way, when there is an I-cache miss, there often are additional instructions in the instruction queue to be processed, thereby not freezing the pipeline.

If there is an I-cache miss, several different scenarios are possible. First, the instruction-prefetch buffers are examined to see whether the requested instructions are there, and, if so, logic steers these instructions into the pipeline as though they came from the I-cache and will also write the critical sector into the I-cache. If the instructions are not found in the instruction-prefetch buffer, a demand-fetch reload request is sent to the L2. The L2 processes this reload request with high priority. When it is returned from the L2, an attempt will be made to write it into the I-cache.

In addition to these demand-oriented instruction-fetching mechanisms, POWER4 prefetches instruction-cache lines that might soon be referenced into its instruction-prefetch buffer, which is capable of holding four entries of 32 instructions each. The instruction-prefetch logic monitors demand instruction-fetch requests and initiates prefetches for the next one (if there is a hit in the prefetch buffer) or two (if there is a miss in the prefetch buffer) sequential cache lines after verifying that they are not already in the I-cache. When these requests return cache lines, the returned lines are stored in the instruction-prefetch buffer so that they do not pollute the demand-oriented I-cache. The I-cache contains only cache lines that have had a reference to at least one instruction.

Decode, crack, and group formation

As instructions are executed out of order, it is necessary to remember the program order of all instructions in flight. To minimize the logic necessary to track a large number of in-flight instructions, groups of instructions are

formed. The individual groups are tracked through the system. That is, the state of the machine is preserved at group boundaries, not at an instruction boundary within a group. Any exception causes the machine to be restored to the state of the oldest group prior to the exception.

A group contains up to five internal instructions referred to as IOPs. In the decode stages, the instructions are placed sequentially in a group—the oldest instruction is placed in slot 0, the next oldest one in slot 1, and so on. Slot 4 is reserved solely for branch instructions. If required, no-ops are inserted to force the branch instruction to be in the fourth slot. If there is no branch instruction, slot 4 contains a no-op. Only one group of instructions can be dispatched in a cycle, and all instructions in a group are dispatched together. (By *dispatch*, we mean the movement of a group of instructions into the issue queues.) Groups are dispatched in program order. Individual IOPs are issued from the issue queues to the execution units out of program order.

Results are committed when the group completes. A group can complete when all older groups have completed and when all instructions in the group have finished execution. Only one group can complete in a cycle.

For correct operation, certain instructions are not allowed to execute speculatively. To ensure that the instruction executes nonspeculatively, it is not executed until it is the next one to complete. This mechanism is called completion serialization. To simplify the implementation, such instructions form single instruction groups. Examples of completion serialization instructions include loads and stores to guarded space and context-synchronizing instructions such as the move-to-machine-state-register instruction that is used to alter the state of the machine.

In order to implement out-of-order execution, many of the architected registers are renamed, but not all. To ensure proper execution of these instructions, any instruction that sets a nonrenamed register terminates a group.

Internal instructions, in most cases, are architected instructions. However, instructions are sometimes split into one or more internal instructions. To achieve high-frequency operation, we have limited instructions to read at most two registers and write at most one register. (Some floating-point operations do not obey this restriction for performance reasons; for example, the fused multiply and add series of instructions are handled directly in the floating-point unit, though they require three sources.) If this is not the case, the instruction is split to satisfy this requirement. As an example, the load with update instruction that loads one register and increments an index register is split into a load and an add instruction. Similarly, the load multiple word instruction is implemented with multiple load word instructions. With respect to group formation, we differentiate two classes. If an instruction is split into two instructions, such as load with update, that action is considered to be *cracking*. If an instruction is split into more than two IOPs, it is called a *millicoded* instruction. Cracked instructions flow into groups like any other instructions, with the restriction that both IOPs must be in the same group. If both IOPs cannot fit into the current group, the group is terminated and a new group is initiated. The instruction following the cracked instruction may be in the same group as the cracked instruction, assuming there is room in the group. Millicoded instructions always begin a new group. The instruction following the millicoded instruction also initiates a new group.

Table 1 Issue queues.

Queue type	*Entries per queue*	*Number of queues*
Fixed-point and load/store	9	4
Floating-point	5	4
Branch execution	12	1
CR logical	5	2

Instructions performing logical operations on the condition register appear at a lower frequency than other instructions. As such, a single execution unit is dedicated to this function. With the instruction-issue rules described in the next section, this condition requires restricting this class of instructions to only two of the four slots, specifically slots 0 and 1.

Group dispatch and instruction issue

Instruction groups are dispatched into the issue queues one group at a time. As a group is dispatched, control information for the group is stored in the group completion table (GCT). The GCT can store up to 20 groups. The GCT entry contains the address of the first instruction in the group. As instructions finish execution, that information is registered in the GCT entry for the group. Information is maintained in the GCT until the group is retired, i.e., either all of its results are committed, or the group is flushed from the system.

Each instruction slot feeds separate issue queues for the floating-point units, the branch-execution unit, the CR execution unit, the fixed-point execution units, and the load/store execution units. The fixed-point and load/store execution units share common issue queues. **Table 1** summarizes the depth of each issue queue and the number of queues available for each type of queue. For the floating-point issue queues and the common issue queues for the fixed-point and load/store units, the issue queues fed from instruction slots 0 and 3 hold instructions to be executed in one of the execution units, while the issue

Table 2 Rename resources.

Resource type	*Logical size*	*Physical size*
GPRs	32 (36)	80
FPRs	32	72
CRs	8 (9) 4-bit fields	32
Link/count registers	2	16
FPSCR	1	20
XER	4 fields	24

queues fed from instruction slots 1 and 2 feed the other execution unit. The CR execution unit draws its instructions from the CR logical issue queue fed from instruction slots 0 and 1.

Instructions are dispatched into the top of an issue queue. As they are issued from the queue, the remaining instructions move down in the queue. In the case of two queues feeding a common execution unit, the two queues are interleaved. The oldest instruction that has all of its sources set in the common interleaved queue is issued to the execution unit.

Before a group can be dispatched, all resources to support that group must be available. If they are not, the group is held until the necessary resources are available. To successfully dispatch, the following resources are assigned:

- *GCT entry:* One GCT entry is assigned for each group. It is released when the group retires.
- *Issue queue slot:* An appropriate issue queue slot must be available for each instruction in the group. It is released when the instruction in it has successfully been issued to the execution unit. Note that in some cases this is not known until several cycles after the instruction has been issued. As an example, a fixed-point operation dependent on an instruction loading a register can be speculatively issued to the fixed-point unit before it is known whether the load instruction resulted in an L1 data cache hit. Should the load instruction miss in the cache, the fixed-point instruction is effectively pulled back and sits in the issue queue until the data on which it depends is successfully loaded into the register.
- *Rename register:* For each register that is renamed and set by an instruction in the group, a corresponding renaming resource must be available. **Table 2** summarizes the renaming resources available to each POWER4 processor. The renaming resource is released when the next instruction writing to the same logical resource is committed. As a result of cracking and millicode used in forming groups, it is necessary in some situations to use additional logical registers. Four additional GPRs and one additional 4-bit CR field are required. The condition register is architected to be one 32-bit register comprising eight 4-bit fields. Each field is treated internal to the system as a separate register and is renamed. Not all of the fixed-point exception register (XER) is renamed; only four of the XER fields are renamed.
- *Load reorder queue (LRQ) entry:* An LRQ entry must be available for each load instruction in the group. These entries are released when the group completes. The LRQ contains 32 entries.
- *Store reorder queue (SRQ) entry:* An SRQ entry must be available for each store instruction in the group. These entries are released when the result of the store is successfully sent to the L2, after the group completes. The SRQ contains 32 entries.

The operation of the LRQ and the SRQ is described in the section on the load/store unit.

As noted previously, certain instructions require completion serialization. Groups so marked are not issued until that group is the next to complete (i.e., all prior groups have successfully completed). Additionally, instructions that read a nonrenamed register cannot be executed until we are sure that all writes to that register have completed. To simplify the implementation, any instruction that writes to a nonrenamed register sets a switch that is reset when the instruction finishes execution. If the switch is set, this blocks dispatch of an instruction that reads a nonrenamed register. Writes to a nonrenamed register are guaranteed to be in program order by making them completion-serialization operations.

Since instruction progression through the machine is tracked in groups, when a particular instruction within a group must signal an interrupt, this is achieved by flushing all of the instructions (and results) of the group and then redispatching the instructions into single instruction groups. A similar mechanism is used to ensure that the fixed-point exception register summary overflow bit is correctly maintained.

Load/store unit operation

The load/store unit requires special attention in an out-of-order execution machine in order to ensure memory consistency. Since we cannot be sure that the result of a store operation will be committed at the time it is executed, a special mechanism is employed. Associated with each SRQ entry is a store data queue (SDQ) entry. The SDQ entry maintains the desired result to be stored until the group containing the store instruction is committed. Once committed, the data maintained in the SDQ is written to the caches. Additionally, three particular hazards must be avoided:

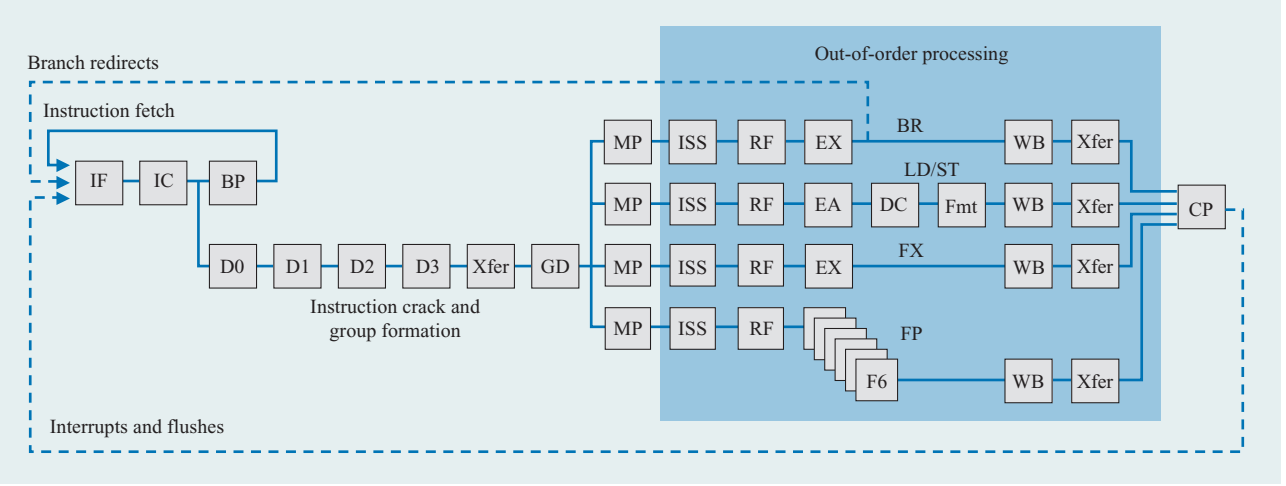

Figure 4

POWER4 instruction execution pipeline.

- *Load hit store:* A younger load that executes before an older store to the same memory location has written its data to the caches must retrieve the data from the SDQ. As loads execute, they check the SRQ to see whether there is any older store to the same memory location with data in the SDQ. If one is found, the data is forwarded from the SDQ rather than from the cache. If the data cannot be forwarded (as is the case if the load and store instructions operate on overlapping memory locations and the load data is not the same as or contained within the store data), the group containing the load instruction is *flushed*; that is, it and all younger groups are discarded and refetched from the instruction cache. If we can tell that there is an older store instruction that will write to the same memory location but has yet to write its result to the SDQ, the load instruction is *rejected* and reissued, again waiting for the store instruction to execute.
- *Store hit load:* If a younger load instruction executes before we have had a chance to recognize that an older store will be writing to the same memory location, the load instruction has received stale data. To guard against this, as a store instruction executes it checks the LRQ; if it finds a younger load that has executed and loaded from memory locations to which the store is writing, the group containing the load instruction and all younger groups are flushed and refetched from the instruction cache. To simplify the logic, all groups following the store are flushed. If the offending load is in the same group as the store instruction, the group is flushed, and all instructions in the group form single-instruction groups.
- *Load hit load:* Two loads to the same memory location must observe the memory reference order and prevent a store to the memory location from another processor between the intervening loads. If the younger load obtains old data, the older load must not obtain new data. This requirement is called *sequential load consistency*. To guard against this, LRQ entries for all loads include a bit which, if set, indicates that a snoop has occurred to the line containing the loaded data for that entry. When a load instruction executes, it compares its load address against all addresses in the LRQ. A match against a younger entry which has been snooped indicates that a sequential load consistency problem exists. To simplify the logic, all groups following the older load instruction are flushed. If both load instructions are in the same group, the flush request is for the group itself. In this case, each instruction in the group when refetched forms a single-instruction group in order to avoid this situation the second time around.

Instruction execution pipeline

Figure 4 shows the POWER4 instruction execution pipeline for the various pipelines. The IF, IC, and BP cycles correspond to the instruction-fetching and branch-prediction cycles. The D0 through GD cycles are the cycles during which instruction decode and group formation occur. The MP cycle is the mapper cycle, in which all dependencies are determined, resources assigned, and the group dispatched into the appropriate issue queues. During the ISS cycle, the IOP is issued to the appropriate execution unit, reads the appropriate

register to retrieve its sources during the RF cycle, and executes during the EX cycle, writing its result back to the appropriate register during the WB cycle. At this point, the instruction has finished execution but has not yet been completed. It cannot complete for at least two more cycles, the Xfer and CP cycle, assuming that all older groups have completed and all other instructions in the same group have also finished.

Instructions waiting in the instruction queue after being fetched from the instruction cache wait prior to the D1 cycle. This is the case if instructions are fetched (up to eight per cycle) faster than they can be formed into groups. Similarly, instructions can wait prior to the MP cycle if resources are not available to dispatch the entire group into the issue queues. Instructions wait in the issue queues prior to the ISS cycle. Similarly, they can wait to complete prior to the CP cycle.

Though not shown, the CR logical execution unit (the unit responsible for executing logical operations on the condition register) is identical to the fixed-point execution pipeline, shown as the FX pipeline in the figure. The branch-execution-unit pipeline is shown as the BR pipeline. If a branch instruction is mispredicted with respect to either direction or target, there is at least a 12-cycle branch-mispredict penalty, depending on how long the mispredicted branch had to wait to be issued.

The pipeline for the two load/store units is identical and is shown as the LD/ST pipeline in Figure 4. After accessing the register file, load and store instructions generate the effective address in the EA cycle. The DERAT and, for load instructions, the data cache directory and the data cache, are all accessed during the DC cycle. If a DERAT miss should occur, the instruction is rejected; i.e., it is kept in the issue queue. Meanwhile, a request is made to the TLB to reload the DERAT with the translation information. The rejected instruction is reissued again a minimum of seven cycles after it was first issued. If the DERAT still does not contain the translation information, the instruction is rejected again. This process continues until the DERAT is reloaded. If a TLB miss occurs (i.e., we do not have translation information in the TLB), the translation is initiated speculatively. However, the TLB is not updated until we are certain that the instruction failing translation will be executed. To ensure that this occurs, the TLB updates are held off until the group that contains the instruction failing translation is the next group to complete. Two different page sizes, 4 KB and 16 MB, are supported. This provides a performance benefit for applications requiring large memory, as is the case in many technical applications and database applications that manage an in-memory pool for indexes or caching data.

In the case of loads, if the directory indicates that the L1 data cache contains the cache line, the requested bytes from the returned data are formatted (the fmt cycle) and written into the appropriate register. They are also available for use by dependent instructions during this cycle. In anticipation of a data cache hit, dependent instructions are issued so that their RF cycle lines up with the writeback cycle of the load instructions. If a cache miss is indicated, a request is initiated to the L2 to retrieve the line. Requests to the L2 are stored in the load miss queue (LMQ). The LMQ can hold up to eight requests to the L2. If the LMQ is full, the load instruction missing in the data cache is rejected and reissued again in seven cycles, and the process is repeated. If there is already a request to the L2 for the same line from another load instruction, the second request is merged into the same LMQ entry. If this is the third request to the same line, the load instruction is rejected, and processing continues as above. All reloads from the L2 check the LMQ to see whether there is an outstanding request yet to be honored against a just-returned line. If there is, the requested bytes are forwarded to the register to complete the execution of the load instruction. After the line has been reloaded, the LMQ entry is freed for reuse.

In the case of store instructions, rather than write data to the data cache, the data is stored in the SDQ as described above. Once the group containing the store instruction is completed, an attempt is made to write the data into the data cache. If the cache line containing the data is already in the L1 data cache, the changed data is written to the data cache. If it is not, the line is not reloaded from the L2. In both cases, the changed data is written to the L2. The coherency point for POWER4 is the L2 cache. Additionally, all data in the L1 data cache are also in the L2 cache. If data has to be cast out of the L2, the line is marked invalid in the L1 data cache if it is resident there.

The pipeline for the two fixed-point execution units is shown as the FX pipe in Figure 4. Two fixed-point instructions, with one dependent on the other, cannot issue on successive cycles. There must be at least one dead cycle between their issue cycles. However, another instruction may issue in this cycle.

The pipeline for the two floating-point execution units is shown as the FP pipe in the figure. Floating-point instructions require six execution cycles. However, both pipelines are fully piped; that is, two instructions can be issued to the floating-point pipes each cycle. A floating-point instruction dependent on a prior floating-point instruction cannot issue within six cycles of the instruction on which it is dependent. However, as is the case with the other execution units, other floating-point instructions may issue during this period.

Storage hierarchy

The POWER4 storage hierarchy consists of three levels of cache and the memory subsystem. The first and second levels of the hierarchy are actually on the POWER4 chip. The directory for the third-level cache, the L3, is on the chip, but the actual cache is off-chip. **Table 3** shows capacities and organization of the various levels of the hierarchy on a per-chip basis.

L1 caches

The L1 instruction cache is single-ported, capable of either one 32-byte read or one 32-byte write each cycle. The store through L1 data cache is triple-ported, capable of two 8-byte reads and one 8-byte write per cycle with no blocking. L1 data-cache reloads are 32 bytes per cycle. The L1 caches are parity-protected. A parity error detected in the L1 instruction cache forces the line to be invalidated and reloaded from the L2. Errors encountered in the L1 data cache are reported as a synchronous machine-check interrupt. To support error recovery, the machine-check interrupt handler is implemented in system-specific firmware code. When the interrupt occurs, the firmware saves the processor-architected states and examines the processor registers to determine the recovery and error status. If the interrupt is recoverable, the system firmware removes the error by invalidating the L1 data-cache line and incrementing an error counter. If the L1 data-cache error counter is greater than a predefined threshold, which is an indication of a solid error, the system firmware disables the failing portion of the L1 data cache. The system firmware then restores the processor-architected states and "calls back" the operating system machine-check handler with the "fully recovered" status. The operating system checks the return status from firmware and resume execution. With the L1 data-cache line invalidated, data is now reloaded from the L2. All data stored in the L1 data cache is available in the L2 cache, guaranteeing no data loss.

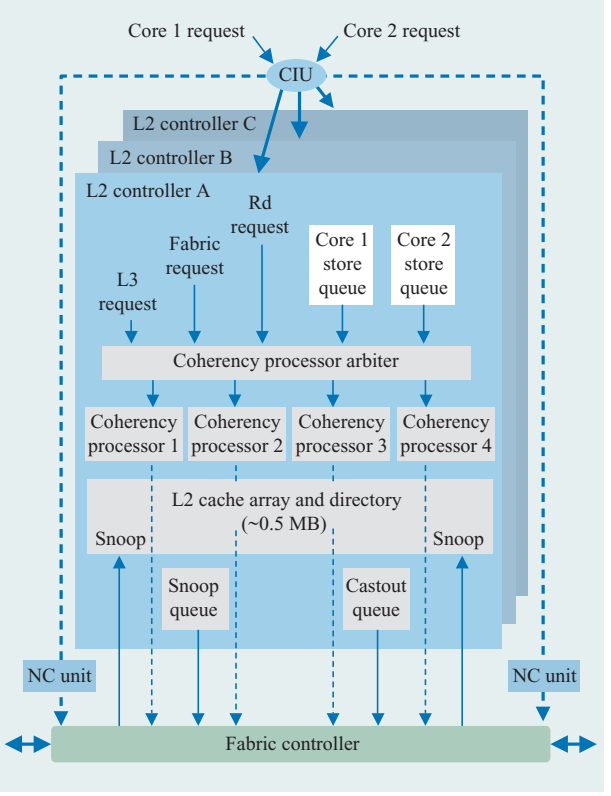

Figure 5

L2 logical view.

Data in the L1 cache can be in one of two states: I (the invalid state, in which the data is invalid) or V (the valid state, in which the data is valid).

L2 cache

The unified second-level cache is shared across the two processors on the POWER4 chip. **Figure 5** shows a logical view of the L2 cache. The L2 is implemented as three identical slices, each with its own controller. Cache lines are hashed across the three controllers.

Table 3 Storage hierarchy organization and size.

Component	*Organization*	*Capacity per chip*
L1 instruction cache	Direct map, 128-byte line managed as four 32-byte sectors	128 KB (64 KB per processor)
L1 data cache	Two-way, 128-byte line	64 KB (32 KB per processor)
L2	Eight-way, 128-byte line	~1.5 MB
L3	Eight-way, 512-byte line managed as four 128-byte sectors	32 MB
Memory	—	0–16 GB

Each slice contains four SRAM partitions, each capable of supplying 16 bytes of data every other cycle. The four partitions can supply 32 bytes per cycle, taking four consecutive cycles to transfer a 128-byte line to the processor. The data arrays are ECC-protected (single-error correct, double-error detect). Both wordline and bitline redundancy are implemented.

The L2 cache directory is implemented in two redundant eight-way set-associative parity-protected arrays. The redundancy, in addition to providing a backup capability, also provides two nonblocking read ports to permit snoops to proceed without causing interference to load and store requests.

A pseudo-LRU replacement algorithm is implemented as a standard 7-bit tree structure.

Since the L1 is a store-through design, store requests to the L2 are at most eight bytes per request. The L2 implements two four-entry 64-byte queues for gathering individual stores and minimizing L2 requests for stores.

The majority of control for L2 cache management is handled by four coherency processors in each controller. A separate coherency processor is assigned to handle each request to the L2. Requests can come from either of the two processors (for either an L1 data-cache reload or an instruction fetch) or from one of the store queues. Each coherency processor has associated with it a cast-out processor to handle deallocation of cache lines to accommodate L2 reloads on L2 misses. The coherency processor does the following:

- Controls the return of data from the L2 (hit) or from the fabric controller (miss) to the requesting processor via the CIU.
- Updates the L2 directory as needed.
- Issues fabric commands for L2 misses on fetch requests and for stores that do not hit in the L2 in the M, Me, or Mu state (described below).
- Controls writing into the L2 when reloading because of fetch misses in the L2 or when accepting stores from the processors.
- Initiates back-invalidates to a processor via the CIU resulting from a store from one processor that hits a cache line marked as resident in the other processor's L1 data cache.

Included in each L2 controller are four snoop processors responsible for managing coherency operations snooped from the fabric. When a fabric operation hits on a valid L2 directory entry, a snoop processor is assigned to take the appropriate action. Depending on the type of operation, the inclusivity bits in the L2 directory, and the coherency state of the cache line, one or more of the following actions may result:

- Sending a back-invalidate request to the processor(s) to invalidate a cache line in its L1 data cache.
- Reading the data from the L2 cache.
- Updating the directory state of the cache line.
- Issuing a push operation to the fabric to write modified data back to memory.
- Sourcing data to another L2 from this L2.

In addition to dispatching a snoop processor, the L2 provides a snoop response to the fabric for all snooped operations. When a fabric operation is snooped by the L2, the directory is accessed to determine whether the targeted cache line is resident in the L2 cache and, if so, its coherency state. Coincident with the snoop directory lookup, the snooped address is compared with the addresses of any currently active coherency, cast-out, and snoop processors to detect address-collision scenarios. The address is also compared to the per-processor reservation address registers. On the basis of this information, the snoop response logic determines the appropriate snoop response to send back.

The L2 cache controller also acts as the reservation station for the two processors on the chip in support of the load [double] word and reserve indexed (lwarx/ldarx) and the store [double] word conditional (stwcx/stdcx) instructions. One address register for each processor is used to hold the reservation address. The reservation logic maintains a reservation flag per processor to indicate when a reservation is set. The flag is set when a lwarx or ldarx instruction is received from the processor; it is reset when certain invalidating type operations are snooped, including a store to the reservation address from other processors in the system, or when a stwcx or stdcx instruction succeeds. (A successful store occurs if the reservation flag was not reset by another operation. The success or failure of the conditional store instruction is communicated to the program by setting a bit in the condition register.)

The L2 cache implements an enhanced version of the MESI coherency protocol, supporting seven states as follows:

- *I (invalid state):* The data is invalid. This is the initial state of the L2 entered from a power-on reset or a snoop invalidate hit.
- *SL (shared state, can be source to local requesters):* The data is valid. The cache line may also be valid in other L2 caches. From this state, the data can be sourced to another L2 on the same module via intervention. This state is entered as a result of a processor L1 data-cache load request or instruction-fetch request that misses in the L2 and is sourced from another cache or from memory when not in other L2s.

- *S (shared state):* The data is valid. The cache line may also be valid in other L2 caches. In this state, the data cannot be sourced to another L2 via intervention. This state is entered when a snoop-read hit from another processor on a chip on the same module occurs and the data and tag were in the SL state.
- *M (modified state):* The data is valid. The data has been modified and is exclusively owned. The cache line cannot be valid in any other L2. From this state the data can be sourced to another L2 in a chip on the same or remote module via intervention. This state results from a store operation performed by one of the processors on the chip.
- *Me (exclusive state):* The data is valid. The data is not considered modified but is exclusive to this L2. The cache line cannot be valid in any other L2. Cast-out of an Me line requires only invalidation of the tag; i.e., data need not be written back to memory. This state is entered as a result of one of the processors on the chip asking for a reservation via the lwarx or ldarx instruction when data is sourced from memory or for a cache line being prefetched into the L2 that was sourced from memory. (Sourcing data from the L3 in the O state is equivalent to sourcing it from memory.)
- *Mu (unsolicited modified state):* The data is valid. The data is considered to have been modified and is exclusively owned. The cache line cannot be valid in any other L2. This state is entered as a result of one of the processors on the chip asking for a reservation via the lwarx or ldarx instruction when data is sourced from another L2 in the M state or for a cache line being prefetched into the L2 that was sourced from another L2 in the M state.
- *T (tagged state):* The data is valid. The data is modified with respect to the copy in memory. It has also been sourced to another cache; i.e., it was in the M state at some time in the past, but is not currently exclusively owned. From this state, the data will not be sourced to another L2 via intervention until the combined response is received and it is determined that no other L2 is sourcing data (that is, if no L2s have the data in the SL state. This state is entered when a snoop-read hit occurs while in the M state.

The L2 state is maintained in the L2 directory. **Table 4** summarizes the L2 states and the possible L1 data-cache state as well as possible states in other L2s. The directory also includes bits to indicate whether or not the data may be contained in one or both of the processor's L1 data caches. Whenever a processor requests data to be loaded into its L1 data cache, a bit corresponding to that processor is set. This bit is not set for instruction fetches. This indication is imprecise because it is not reset if the data is replaced by the L1.

Table 4 Valid L2 states.

L2 state	*L1 data cache*	*State in other L2s*
I	I	Any
S_L	I, V	I, S, S_L, T
S	I, V	I, S, T
M, Me, or Mu	I, V	I
T	I, V	I, S, S_L

Included within the L2 subsystem are two noncacheable units (NCU), one per processor, labeled NC units in Figure 5. The NCUs handle noncacheable loads and stores, as well as cache and synchronization operations. (Because the storage model implemented is a weakly consistent one, it is the programmer's responsibility to explicitly code memory-synchronization instructions in those cases where the order of memory operations, as seen by another processor, must be enforced.) Each NCU is partitioned into two parts: the NCU master and the NCU snooper. The NCU master handles requests originating from processors on the chip, while the NCU snooper handles the snooping of translation lookaside buffer invalidate entry (tlbie) and instruction cache block invalidate (icbi) operations from the fabric.

The NCU master includes a four-deep FIFO queue for handling cache-inhibited stores, including memory-mapped I/O store operations, and cache and memory barrier operations. It also contains a one-deep queue for handling cache-inhibited load operations.

The return of data for a noncacheable load operation is via the L2 controller, using the same reload buses as for cacheable load operations. Cache-inhibited stores are routed through the NCU in order to preserve execution ordering of noncacheable stores with respect to one another.

Cache and synchronization operations originating in a processor on the chip are handled in a manner similar to cache-inhibited stores, except that they do not have any data associated with them. These operations are issued to the fabric. Most will be snooped by an L2 controller. Included in this category are the icbi, tlbie, translation lookaside buffer synchronize (tlbsync), enforce in-order execution of I/O (eieio), synchronize (sync), page table entry synchronize (ptesync), lsync, data cache block flush (dcbf), data cache block invalidate (dcbi) instructions, and a processor acknowledgment that a snooped TLB has completed.

The NCU snooper snoops icbi and tlbie operations from the fabric, propagating them upstream to the processors. These snoops are sent to the processor via the reload buses of the L2 controller. It also snoops sync, ptesync,

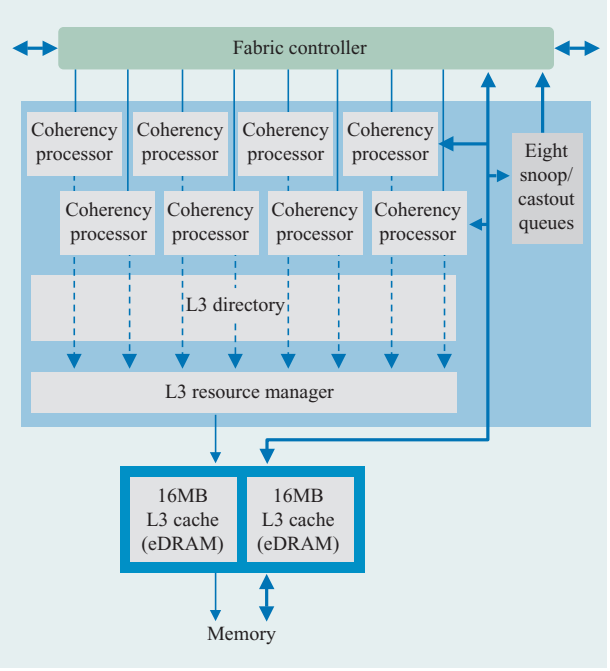

Figure 6

L3 logical view.

Table 5 Mapping of real address bits to access L3 depending on L3 size.

Logical L3 size	*L3 index*	*L3 tag*	*L3 chip select*	*L3 quadrant select*
32 MB	51:52, 42:50	22:41	—	53:54
64 MB	51, 41:50	22:40	54	52:53
128 MB	40:50	22:39	53:54	51:52

lsync, and eieio. These are snooped because they may have to be retried because of an icbi or TLB that has not yet completed to the same processor.

L3 cache

Figure 6 shows a logical view of the L3 cache. The L3 consists of two components, the L3 controller and the L3 data array. The L3 controller is located on the POWER4 chip and contains the tag directory as well as the queues and arbitration logic to support the L3 and the memory behind it. The data array is stored in two 16MB eDRAM chips mounted on a separate module. A separate memory controller can be attached to the back side of the L3 module.

To facilitate physical design and minimize bank conflicts, the embedded DRAM on the L3 chip is organized as eight banks at 2 MB per bank, with banks grouped in pairs to divide the chip into four 4MB quadrants. The L3 controller is also organized in quadrants. Each quadrant contains two coherency processors to service requests from the fabric, perform any L3 cache and/or memory accesses, and update the L3 tag directory. Additionally, each quadrant contains two processors to perform the memory cast-outs, invalidate functions, and DMA writes for I/O operations. Each pair of quadrants shares one of the two L3 tag directory SRAMs.

The L3 cache is eight-way set-associative, organized in 512-byte blocks, with coherence maintained on 128-byte sectors for compatibility with the L2 cache. Five coherency states are supported for each of the 128-byte sectors, as follows:

- *I (invalid state):* The data is invalid.
- *S (shared state):* The data is valid. In this state, the L3 can source data only to L2s for which it is caching data.
- *T (tagged state):* The data is valid. The data is modified relative to the copy stored in memory. The data may be shared in other L2 or L3 caches.
- *Trem (remote tagged state):* This is the same as the T state, but the data was sourced from memory attached to another chip.
- *O (prefetch data state):* The data in the L3 is identical to the data in memory. The data was sourced from memory attached to this L3. The status of the data in other L2 or L3 caches is unknown.

Each L3 coherency processor supports one random cache or memory access. For sequential accesses, the L3 coherency processors can support up to four concurrent load/store requests within a 512-byte L3 cache block. This allows the L3 to support increased cache and memory throughput for many common technical workloads to take advantage of the bandwidth capability available with the high-speed buses in POWER4 systems.

The L3 is designed to be used as a standalone 32MB L3 cache, or to be combined with other L3s on the same processor module in pairs or groups of four to create a larger, address-interleaved L3 cache of 64 MB or 128 MB. Combining L3s into groups not only increases the L3 cache size, but also scales the available L3 bandwidth. When combined into groups, L3s and the memory behind them are interleaved on 512-byte granularity. The fabric bus controller selects the quadrant of the L3 to which a particular real address is mapped, and the selected L3 controller adjusts the mapping from real address to L3 index and tag to account for the increase in the effective cache size. **Table 5** shows the mapping of real address bits to L3 index and tag, as well as the algorithm for routing an address to the corresponding L3 controller and quadrant. The custom address flow logic is optimized for the 128MB combined case. To handle the index/tag

adjustment for the smaller L3 cache sizes, the appropriate bits are swapped as the L3 controller receives an address from the fabric bus controller. This approach causes the index bits to appear in a nonintuitive order, but avoids the need for the custom address flow logic to shift all of the address bits to make this adjustment. All address-bit ranges in Table 5 assume that the full 42-bit address is denoted as bits 22:63. Bits 55:56 are the sector ID bits, and bits 57:63 are the offset within the 128-byte coherence granule.

The L3 caches data, either from memory that resides beneath it or from elsewhere in the system, on behalf of the processors attached to its processor module. When one of its processors issues a load request that misses the L3 cache, the L3 controller allocates a copy of the data in the S (shared) state. Inclusivity with the L1 and L2 is not enforced. Hence, when the L3 deallocates data, it does not invalidate any L1 or L2 copies. The L3 enters the T or Trem state when one of its local L2 caches does a castout from the M or T state. An address decode is performed at snoop time to determine whether the address maps to memory behind the L3 or elsewhere in the system, and this causes the L3 to transition to the T or Trem state as appropriate. This design point was chosen to avoid the need for a memory address-range decode when the L3 performs a cast-out operation. The L3 can use the T/Trem distinction to determine whether the data can be written to the attached memory controller, or whether the cast-out operation must be issued as a fabric bus transaction.

When in the T or Trem state, the L3 sources data to any requestor in the system. However, when in the S state, the L3 will source data only to its own L2s. This minimizes data traffic on the buses between processor modules, since, whenever possible, data is sourced by an L3 cache on the requesting processor module. When in the O state, the L3 sources data to any requestor using the same rules that determine when it is permitted to send data from its attached memory controller; i.e., no cache is sourcing data and no snooper retried the request.

The L3 tag directory is ECC-protected to support single-bit error correct and double-bit error detect. Uncorrectable errors result in a system checkstop. If a directory access results in a correctable error, the access is stalled while the error is corrected. After correction, the original access takes place. When an error is corrected, a recovered attention message is sent to the service processor for thresholding purposes.

The L3 address, memory address, and control buses have parity bits for single-bit error detection. The L3 and memory data buses, as well as the L3-cache-embedded DRAMs, have ECC to support single-bit error correct and double-bit error detect. Uncorrectable errors are flagged and delivered to the requesting processor with an error indication, resulting in a machine-check interrupt. Correctable errors are corrected in-line, and a recovered attention message is sent to the service processor for thresholding purposes.

The L3 supports cache-line delete. The cache-line delete function is used to mask stuck faults in the L3-cache-embedded DRAMs. Line-delete control registers allow the service processor to specify L3 index values for which a particular member should not be used. When the L3 controller snoops a request that matches a specified L3 index, it masks the tag-directory compare for the member in question. The replacement algorithm also avoids the deleted member when choosing a victim in the specified congruence class. Cache-line delete can be invoked at IPL time on the basis of results of power-on diagnostic testing, or it can be enabled dynamically because of a fault detected at run time.

If an L3 tag directory develops a stuck fault, or L3-cache-embedded DRAMs develop more stuck faults than can be handled with the line-delete control registers, the L3 cache on the failing processor chip can be reconfigured and logically removed from the system without removing other L3 caches in the system and without reconfiguring the memory attached to that L3. Memory accesses continue to pass through the reconfigured L3 module, but that L3 controller no longer performs cache operations.

Memory subsystem

A logical view of the memory subsystem is shown in **Figure 7**. Each POWER4 chip can have an optional memory controller attached behind the L3 cache. Memory controllers are packaged two to a memory card and support two of the four processor chips on a module. A module can attach a maximum of two memory cards. Memory controllers can have either one or two ports to memory.

The memory controller is attached to the L3 eDRAM chips, with each chip having two 8-byte buses, one in each direction, to the data interface in the memory controller. These buses operate at one-third processor speed using the synchronous wave pipeline interface [21] to operate at high frequencies.

Each port to memory has four 4-byte bidirectional buses operating at a fixed frequency of 400 MHz connecting load/store buffers in the memory controller to four system memory interface (SMI) chips used to read and write data from memory. When two memory ports are available, they each work on 512-byte boundaries. The memory controller has a 64-entry read command queue, a 64-entry write command queue, and a 16-entry write cache queue.

The memory is protected by a single-bit error correct, double-bit error detect ECC. Additionally, memory scrubbing is used in the background to find and correct soft errors. Each memory extent has an extra DRAM to allow for transparent replacement of one failing DRAM

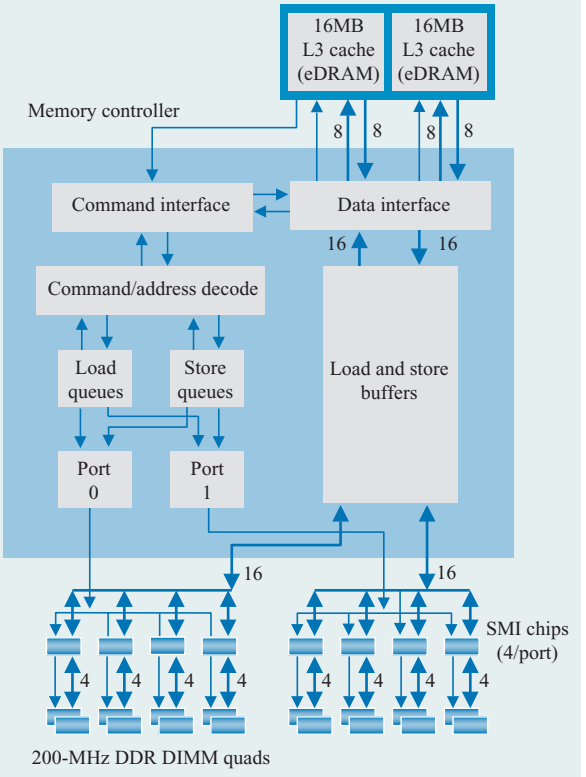

Figure 7

Memory subsystem logical view.

per group of four DIMMs using chip-kill technology. Redundant bit steering is also employed.

Hardware data prefetch

POWER4 systems employ hardware to prefetch data transparently to software into the L1 data cache. When load instructions miss sequential cache lines, either ascending or descending, the prefetch engine initiates accesses to the following cache lines before being referenced by load instructions. In order to ensure that the data will be in the L1 data cache, data is prefetched into the L2 from the L3 and into the L3 from memory. **Figure 8** shows the sequence of prefetch operations. Eight such streams per processor are supported.

Once a sequential reference stream is recognized, whenever a load instruction initiates a request for data in a new cache line, the prefetch engine begins staging the next sequential line into the L1 data cache from the L2. At the same time, it initiates a request to the L3 to stage a line into the L2. However, since latencies to load the L2 from the L3 are longer than the latency to load the L1 from the L2, rather than prefetch the second cache line, the fifth is prefetched, as shown in Figure 8. Prior references, or the initial ramp-up on stream initiation, has already staged the second through fourth lines from the L2 to the L1 data cache. Similarly, a line is replaced in the L3 from memory. To minimize processing required to retrieve data from memory into the L3, a 512-byte line is prefetched. This has to be done only every fourth line referenced. In the case shown in the figure, lines 17 through 20 are prefetched from memory to the L3.

Because memory references are based on real addresses, whenever a page boundary is crossed, the prefetching must be stopped, since we do not know the real address of the next page. To reduce the performance impact, POWER4 implements two page sizes, 4 KB and 16 MB. In addition to allowing the prefetch to continue for longer streams, it saves translation time. This is especially useful for technical applications, where it is common to sequentially reference large amounts of data.

To guard against prematurely initiating a data-prefetch stream by the hardware, POWER4 ramps up the prefetches slowly, requiring an additional four sequential cache misses to occur before all of the cache lines in the entire sequence are in various stages of prefetch to the L1 data cache. However, software can often tell that a prefetch stream should be initiated. Toward this end, the data cache block touch (dcbt) instruction has been extended, using a previously reserved bit to indicate to the hardware that a prefetch stream should be installed immediately without waiting for confirmation.

Special logic to implement data prefetching exists in the processor load/store unit (LSU) and in the L2 and L3. The direction to prefetch, up or down, is determined by the actual load address within the line that causes the cache miss. If the load address is in the lower portion of the line, the guessed direction is up. If the load address is in the upper portion of the line, the guessed direction is down. The prefetch engine initiates a new prefetch when it detects a reference to the line it guessed will be used. If the initial guess on the direction is not correct, the subsequent access will not confirm to the prefetch engine that it had a stream. The incorrectly initiated stream will eventually be deallocated, and the corrected stream will be installed as a new stream.

Interconnecting chips to form larger SMPs

The basic building block is a multichip module (MCM) with four POWER4 chips to form an eight-way SMP. Multiple MCMs can be further interconnected to form 16-, 24-, and 32-way SMPs.

Four-chip, eight-way SMP module

Figure 9 shows the logical interconnection of four POWER4 chips across four logical buses to form an eight-way SMP. Each chip writes to its own bus, arbitrating

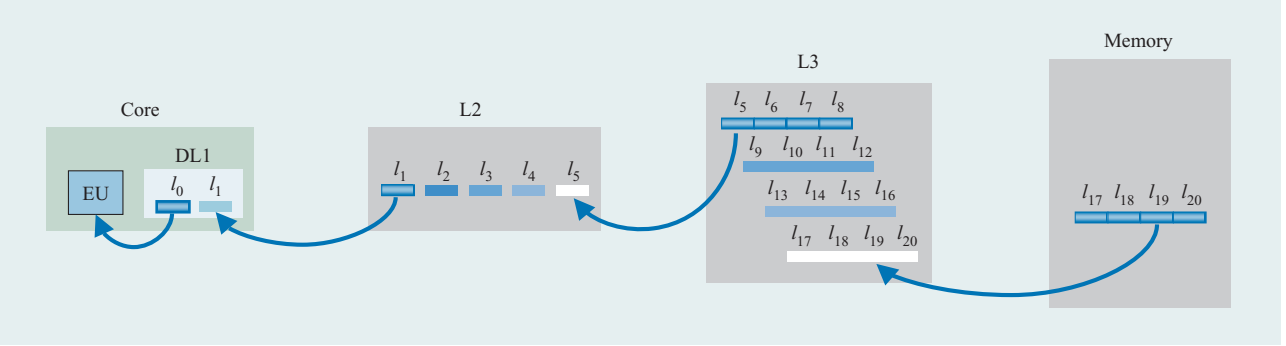

Figure 8

POWER4 hardware data prefetch.

among the L2, I/O controller, and L3 controller for the bus. Each of the four chips snoops all of the buses, and if it finds a transaction that it must act on, it takes the appropriate action. Requests for data from an L2 are snooped by all chips to see a) whether it is in their L2 and in a state that permits sourcing it from the holding chip's L2 to the requesting chip, or b) whether it is in its L3 or in memory behind its L3 cache based on the real address of the request. If it is, the sourcing chip returns the requested data to the requesting chip on its bus.

The interconnection topology appears like a bus-based system from the perspective of a single chip. From the perspective of the module, it appears like a switch.

Multiple module interconnect

Figure 10 shows the interconnection of multiple four-chip MCMs to form larger SMPs. One to four MCMs can be interconnected. When interconnecting multiple MCMs, the intermodule buses act as repeaters, moving requests and responses from one module to another module in a ring topology. As with the single MCM configuration, each chip always sends requests/commands and data on its own bus but snoops all buses.

L3 memory configurations

As noted earlier, each MCM can have from zero to two memory cards. In the case of two memory cards, there is no requirement that they be of equal size. In the case of no memory cards or two equal-size memory cards connected to an MCM, the four L3s attached to the module act as a single 128MB L3. In a single MCM system, each L3 caches data sourced from the memory attached behind its L3 cache. In the case of multiple MCMs and data being sourced from memory attached to another module, an attempt is made to cache the returned data on the requesting module. The particular L3 chosen is the L3 attached to the chip controlling the bus on which

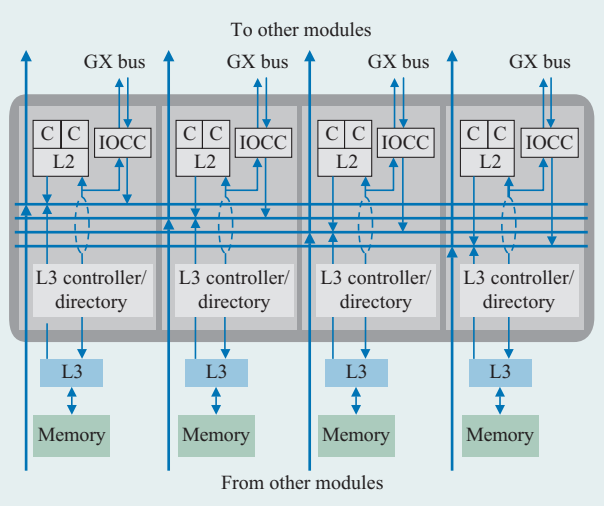

Figure 9

POWER4 multichip module with four chips forming an eight-way SMP.

the data is returned. However, if the L3 is busy servicing requests, the line is not cached. Also, data is not cached on the sourcing module if it is being sourced to a chip on another module.

If one memory card or two memory cards of unequal size are attached to a module, the L3s attached to the module function as two 64MB L3s. The two L3s that act in concert are the L3s that would be in front of the memory card. (Note that one memory card is attached to two chips.) The caching of requests to remote modules described above functions in this case in a comparable manner, with the exception that the two L3s acting as a single L3 are considered to logically form a module boundary (for caching purposes).

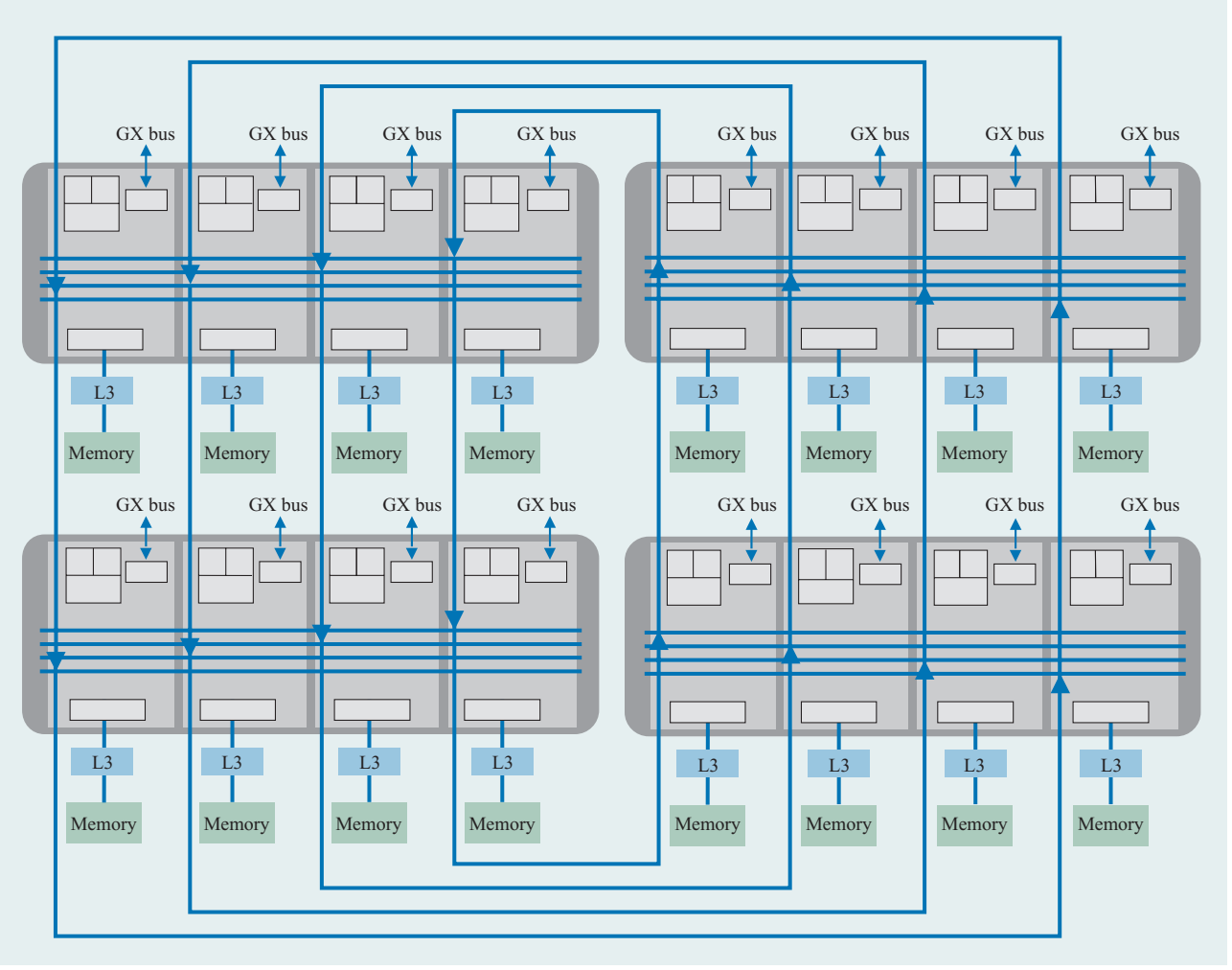

Figure 10

Multiple POWER4 multichip module interconnection.

I/O structure

Figure 11 shows the I/O structure in POWER4 systems. Attached to a POWER4 GX bus is a remote I/O (RIO) bridge chip. This chip transmits the data across two 1-byte-wide RIO buses to PCI host bridge (PHB) chips. Two separate PCI buses attach to PCI–PCI bridge chips that further fan the data out across multiple PCI buses. When multiple nodes are interconnected to form clusters of systems, the RIO bridge chip is replaced with a chip that connects with the switch. This provides increased bandwidth and reduced latency over switches attached via the PCI interface.

System balance

POWER4 systems are designed to deliver balanced performance. As an example, as additional chips and MCMs are added to form larger SMP systems, additional resources are added, as can be seen from Figure 10. With the addition of each pair of POWER4 chips, the ability to add a memory card is provided. In addition to memory capacity, memory bandwidth is increased. Each additional POWER4 chip provides additional L3 resource.

All buses interconnecting POWER4 chips, whether or not on- or off-module, operate at half processor speed. As future technology is exploited, allowing chip size to decrease and operating frequencies to increase, system balance is maintained, since bus speeds are no longer fixed but are geared to processor frequency.

The multi-MCM configuration provides a worst-case memory access latency of slightly greater than 10% more than the best-case memory access latency maintaining the flat memory model, simplifying programming.

The eight-way MCM is the building block for the system. It is only available with four chips, each with its attached L3. A single processor on a chip has all of the L3 resources attached to the module, and the full L2 on the chip itself. If this processor is the only processor executing, it exhibits extremely good performance. If only one chip of the four on the module is active, the situation is similar, though both processors now share a common L2. They both have full access to all of the L3s attached to the module. When analyzing measurements comparing one-way to two-way to four-way to eight-way performance, one must account for the full L3 available in all of these configurations.

Future roadmap

Enhancements to the current POWER4 system in the coming years will take several directions.

We are in the process of leveraging newer technologies to allow us to increase frequency while further decreasing power. We will aggressively increase processor frequencies to the 2+ GHz range while maintaining the system balance offered by our current design.

The current design introduces parallelism throughout the system so as to overcome the increasing memory latencies (in processor cycles) resulting from high-frequency operations. The parallelism allows the processor to continue executing in the presence of cache misses. Future POWER4 systems will continue this design, increasing parallelism and providing larger caches.

We have already invested in ensuring that software can exploit the increased performance levels POWER4 systems will be offering. We will continue making system-level enhancements in order to provide even greater performance increases over time.

Summary

POWER4 development has met our objectives in terms of performance and schedule as defined at the beginning of the project in 1996. Enormous levels of bandwidth and concurrency contribute to superior performance across a broad range of commercial and high-performance computing environments. These unprecedented performance levels are achieved by a total system design that exploits leading IBM technologies. We are well along in developing follow-on systems to the current POWER4 to further enhance its leadership. POWER4 is redefining what is meant by a server and how a server must be designed.

Acknowledgments

An effort like this requires a large design, development, verification, and test team. This project included engineers and programmers from several IBM research and development laboratories. The focal point for the project was in the IBM Server Group Development Laboratory in Austin, Texas, with major contributions from Server Group development laboratories in Poughkeepsie, New York, and Rochester, Minnesota, as well as IBM research laboratories in Yorktown Heights, New York, and Austin, Texas. It is impossible to single out specific contributors, as the list would be too lengthy. Suffice it to say that as authors we represent a large, proud team that have documented the contributions of many individuals.

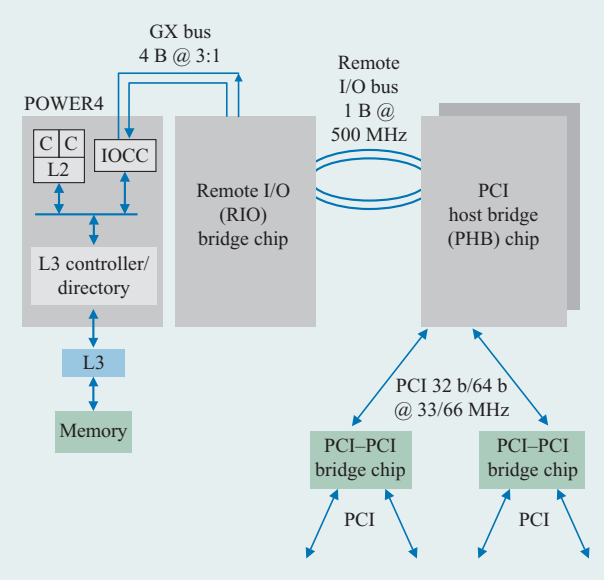

Figure 11

I/O logical view.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of The Open Group.

References

1. H. B. Bakoglu, G. F. Grohoski, and R. K. Montoye, "The IBM RISC System/6000 Processor Hardware Overview," *IBM J. Res. & Dev.* **34,** No. 1, 12–22 (January 1990).
2. John Cocke and V. Markstein, "The Evolution of RISC Technology at IBM," *IBM J. Res. & Dev.* **34,** No. 1, 4–11 (January 1990).
3. S. W. White and S. Dhawan, "POWER2: Next Generation of the RISC System/6000 Family," *IBM J. Res. & Dev.* **38,** No. 5, 493–502 (September 1994).
4. M. T. Vaden, L. J. Merkel, C. R. Moore, T. M. Potter, and R. J. Reese, "Design Considerations for the PowerPC 601 Microprocessor," *IBM J. Res. & Dev.* **38,** No. 5, 605–620 (September 1994).
5. Mark Papermaster, Robert Dinkjan, Michael Mayfield, Peter Lenk, Bill Ciarfella, Frank O'Connell, and Raymond DuPont, "POWER3: Next Generation 64-bit PowerPC Processor Design," IBM White Paper, October 1998, available on the World Wide Web at *http://www-1.ibm.com/servers/eserver/pseries/hardware/whitepapers/power3wp.pdf*.
6. John Borkenhagen and Salvatore Storino, "4th Generation 64-bit PowerPC-Compatible Commercial Processor

Design," IBM White Paper, January 1999, available at *http://www-1.ibm.com/servers/eserver/pseries/hardware/whitepapers/nstar.pdf*.
7. John Borkenhagen and Salvatore Storino, "5th Generation 64-bit PowerPC-Compatible Commercial Processor Design," IBM White Paper, September 1999, available at *http://www-1.ibm.com/servers/eserver/pseries/hardware/whitepapers/pulsar.pdf*.
8. J. M. Borkenhagen, R. J. Eickemeyer, R. N. Kalla, and S. R. Kunkel, "A Multithreaded PowerPC Processor for Commercial Servers," *IBM J. Res. & Dev.* **44,** No. 6, 885–898 (November 2000).
9. F. P. O'Connell and S. W. White, "POWER3: The Next Generation of PowerPC Processors," *IBM J. Res. & Dev.* **44,** No. 6, 873–884 (November 2000).
10. C. May (Ed.), E. M. Silha, R. Simpson, and H. S. Warren, Jr. (Ed.), *The PowerPC Architecture: A Specification for a New Family of RISC Processors*, Morgan Kaufmann Publishers, Inc., San Francisco, 1994.
11. E. Leobandung, E. Barth, M. Sherony, S.-H. Lo, R. Schulz, W. Chu, M. Khare, D. Sadana, D. Schepis, R. Bolam, J. Sleight, F. White, F. Assaderaghi, D. Moy, G. Biery, R. Goldblatt, T.-C. Chen, B. Davari, and G. Shahidi, "High-Performance 0.18 μm SOI CMOS Technology," *IEDM Tech Digest*, pp. 679–682 (1999).
12. Linley Gwennap, "Speed Kills," *Microprocessor Report*, March 8, 1993, p. 3.
13. See for example the results for the SPECint2000 and SPECfp2000 benchmarks available at *www.spec.org*.
14. D. C. Bossen, A. Kitamorn, K. F. Reick, and M. S. Floyd, "Fault-Tolerant Design of the IBM pSeries 690 System Using POWER4 Processor Technology," *IBM J. Res. & Dev.* **46,** No. 1, 77–86 (2002, this issue).
15. J. D. Warnock, J. M. Keaty, J. Petrovick, J. G. Clabes, C. J. Kircher, B. L. Krauter, P. J. Restle, B. A. Zoric, and C. J. Anderson, "The Circuit and Physical Design of the POWER4 Microprocessor," *IBM J. Res. & Dev.* **46,** No. 1, 27–51 (2002, this issue).
16. T.-Y. Yeh and Y. N. Patt, "A Comparison of Dynamic Branch Predictors that Use Two Levels of Branch History," *Proceedings of the 24th Annual ACM/IEEE International Symposium on Microarchitecture*, 1993, pp. 257–266.
17. S. McFarling, "Combining Branch Predictors," *Technical Report TN-36*, Digital Western Research Laboratory, Palo Alto, CA, June 1993.
18. R. E. Kessler, "The Alpha 21264 Microprocessor," *IEEE Micro* **19,** No. 2, 24–36 (March–April 1999).
19. P. Y. Chang, E. Hao, and Y. N. Patt, "Target Prediction for Indirect Jumps," *Proceedings of the 24th Annual International Symposium on Computer Architecture*, Denver, June 1997, pp. 274–283.
20. C. F. Webb, "Subroutine Call/Return Stack," *IBM Tech. Disclosure Bull.* **30,** No. 11, 18–20 (April 1988).
21. Frank Ferraiolo, Edgar Cordero, Daniel Dreps, Michael Floyd, Kevin Gower, and Bradley McCredie, "POWER4 Synchronous Wave-Pipelined Interface," paper presented at the Hot Chips 11 Conference, Palo Alto, August 1999.

Received October 29, 2001; accepted for publication December 27, 2001

Joel M. Tendler *IBM Server Group, 11400 Burnet Road, Austin, Texas 78758 (jtendler@us.ibm.com).* Dr. Tendler is Program Director of Technology Assessment, responsible for assessing emerging technologies for applicability in future eServer iSeries and pSeries product offerings. He has extensive hardware and software design experience in S/390 and RS/6000 systems. Before assuming his current position, he was the lead performance analyst on the POWER4 project. Dr. Tendler received a B.E. degree in electrical engineering from the Cooper Union, and a Ph.D. degree in electrical engineering from Syracuse University. He has seven U.S. patents pending and one published; he has received a First Plateau IBM Invention Achievement Award.

J. Steve Dodson *IBM Server Group, 11400 Burnet Road, Austin, Texas 78758 (jsdodson@us.ibm.com).* Mr. Dodson received his B.S.E.E. degree from the University of Kentucky in 1982 and joined IBM in 1983 in Lexington, Kentucky. While in Lexington, he worked in VLSI component engineering, performing characterization and reliability studies of various vendor SRAMs, EPROMs, and microcontrollers. He moved to Austin in 1987 as a logic designer on the first POWER microprocessors. After serving as the design lead for two I/O host bridge chip development efforts, he moved on to develop two generations of level-2 cache controller chips which were used in the RS/6000 Model F50 server and related products. He then joined the POWER4 team as team lead for the level-2 cache controller unit, and is currently involved in the development effort of a future POWER microprocessor as technical lead for the memory subsystem design. Mr. Dodson was appointed to the position of Senior Technical Staff Member in 2000. He has coauthored numerous patent applications in the areas of SMP cache coherency, cache hierarchy, and system bus protocols.

J. S. Fields, Jr. (Steve) *IBM Server Group, 11400 Burnet Road, Austin, Texas 78758 (sfields@us.ibm.com).* Mr. Fields received a B.S.E.E. degree from the University of Illinois. He has designed I/O, memory, and cache subsystems for servers for 13 years. He was the lead designer on the POWER4 L3 cache, and led the hardware bringup and validation efforts for the POWER4 storage subsystem. Mr. Fields holds nine U.S. patents, with another 69 patents pending.

Hung Le *IBM Server Group, 11400 Burnet Road, Austin, Texas 78758 (hung@us.ibm.com).* Mr. Le is a Senior Technical Staff Member in the POWER4 Development team in Austin, Texas. He joined IBM in 1979 after receiving a B.S. degree in electrical and computer engineering from Clarkson University. He has worked in the development of several ES/9000 mainframe CPUs as well as the POWER3 and POWER4 microprocessors. His technical interest is in superscalar and multithreading design. Mr. Le has received an IBM Outstanding Innovation Award and holds 31 issued patents.

Balaram Sinharoy *IBM Server Group, 522 South Road, Poughkeepsie, New York 12601 (balaram@us.ibm.com).* Dr. Sinharoy is a Senior Engineer in the IBM advanced microprocessor design group, where he is currently the chief scientist and technical lead for a future POWER microprocessor. He designed the branch-prediction unit in POWER4 and led several architectural, design, performance, and verification aspects of the POWER4 design. Before

joining the POWER4 microprocessor development group in 1996, Dr. Sinharoy worked in the IBM VLIW compiler and architecture research group on hardware multithreading. He joined IBM in 1992 after receiving his Ph.D. degree in computer science from Rensselaer Polytechnic Institute. He received the 1992 Robert McNaughton Award as a recognition of his research contributions. His research and development interests include advanced microprocessor design, computer architecture and performance analysis, instruction-level parallel processing, and compiler optimization. Dr. Sinharoy has published numerous articles and received patents in these areas. He has received a Sixth Plateau IBM Invention Achievement Award and a Third Plateau IBM Publication Achievement Award. Dr. Sinharoy is a Senior Member of IEEE and a member of ACM.