IBM Power5 Chip: A Dual-Core Multithreaded Processor

Featuring single- and multithreaded execution, the Power5 provides higher performance in the single-threaded mode than its Power4 predecessor at equivalent frequencies. Enhancements include dynamic resource balancing to efficiently allocate system resources to each thread, software-controlled thread prioritization, and dynamic power management to reduce power consumption without affecting performance.

Ron Kalla
Balaram Sinharoy
Joel M. Tendler
IBM

●●●●●● IBM introduced Power4-based systems in 2001.[1] The Power4 design integrates two processor cores on a single chip, a shared second-level cache, a directory for an off-chip third-level cache, and the necessary circuitry to connect it to other Power4 chips to form a system. The dual-processor chip provides natural thread-level parallelism at the chip level. Additionally, the Power4's out-of-order execution design lets the hardware bypass instructions whose operands are not yet available (perhaps because of an earlier cache miss during register loading) and execute other instructions whose operands are ready. Later, when the operands become available, the hardware can execute the skipped instruction. Coupled with a superscalar design, out-of-order execution results in higher instruction execution parallelism than otherwise possible.

The Power5 is the next-generation chip in this line. One of our key goals in designing the Power5 was to maintain both binary and structural compatibility with existing Power4 systems to ensure that binaries continue executing properly and all application optimizations carry forward to newer systems. With that base requirement, we specified increased performance and other functional enhancements of server virtualization, reliability, availability, and serviceability at both chip and system levels. In this article, we describe the approach we used to improve chip-level performance.

Multithreading

Conventional processors execute instructions from a single instruction stream. Despite microarchitectural advances, execution unit utilization remains low in today's microprocessors. It is not unusual to see average execution unit utilization rates of approximately 25 percent across a broad spectrum of environments. To increase execution unit utilization, designers use thread-level parallelism, in which the physical processor core executes instructions from more than one instruction stream. To the operating system, the physical processor core appears as if it is a symmetric multiprocessor containing two logical processors. There are at least three different methods for handling multiple threads.

In coarse-grained multithreading, only one

thread executes at any instance. When a thread encounters a long-latency event, such as a cache miss, the hardware swaps in a second thread to use the machine's resources, rather than letting the machine remain idle. By allowing other work to use what otherwise would be idle cycles, this scheme increases overall system throughput. To conserve resources, both threads share many system resources, such as architectural registers. Hence, swapping program control from one thread to another requires several cycles. IBM implemented coarse-grained multithreading in the IBM eServer pSeries Model 680.[2]

A variant of coarse-grained multithreading is fine-grained multithreading. Machines of this class execute threads in successive cycles, in round-robin fashion.[3] Accommodating this design requires duplicate hardware facilities. When a thread encounters a long-latency event, its cycles remain unused.

Finally, in simultaneous multithreading (SMT), as in other multithreaded implementations, the processor fetches instructions from more than one thread.[4] What differentiates this implementation is its ability to schedule instructions for execution from all threads concurrently. With SMT, the system dynamically adjusts to the environment, allowing instructions to execute from each thread if possible, and allowing instructions from one thread to utilize all the execution units if the other thread encounters a long-latency event.

The Power5 design implements two-way SMT on each of the chip's two processor cores. Although a higher level of multithreading is possible, our simulations showed that the added complexity was unjustified. As designers add simultaneous threads to a single physical processor, the marginal performance benefit decreases. In fact, additional multithreading might decrease performance because of cache thrashing, as data from one thread displaces data needed by another thread.

Power5 system structure

Figure 1 shows the high-level structures of Power4- and Power5-based systems. The Power4 handles up to a 32-way symmetric multiprocessor. Going beyond 32 processors increases interprocessor communication, resulting in high traffic on the interconnection
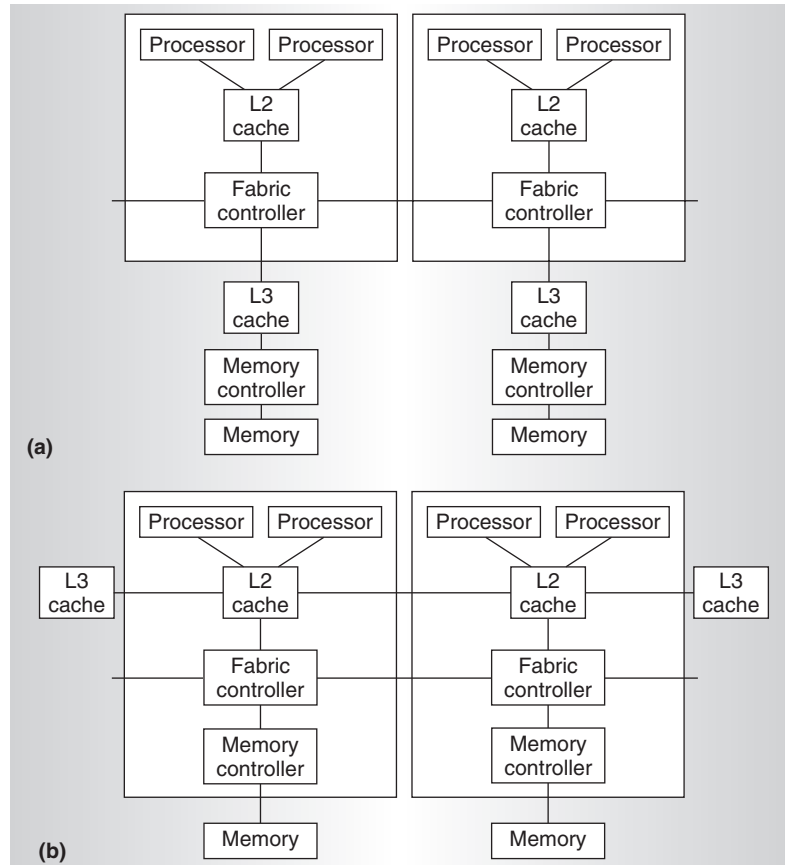

Figure 1. Power4 (a) and Power5 (b) system structures.

fabric. This can cause greater contention and negatively affect system scalability. Moving the level-three (L3) cache from the memory side to the processor side of the fabric lets the Power5 more frequently satisfy level-two (L2) cache misses with hits in the 36-Mbyte off-chip L3 cache, avoiding traffic on the interchip fabric. References to data not resident in the on-chip L2 cache cause the system to check the L3 cache before sending requests onto the interconnection fabric. Moving the L3 cache provides significantly more cache on the processor side than previously available, thus reducing traffic on the fabric and allowing Power5-based systems to scale to higher levels of symmetric multiprocessing. Initial Power5 systems support 64 physical processors.

The Power4 includes a 1.41-Mbyte on-chip L2 cache. Power4+ chips are similar in design to the Power4 but are fabricated in 130-nm technology rather than the Power4's 180-nm technology. The Power4+ includes a 1.5-Mbyte on-chip L2 cache, whereas the Power5
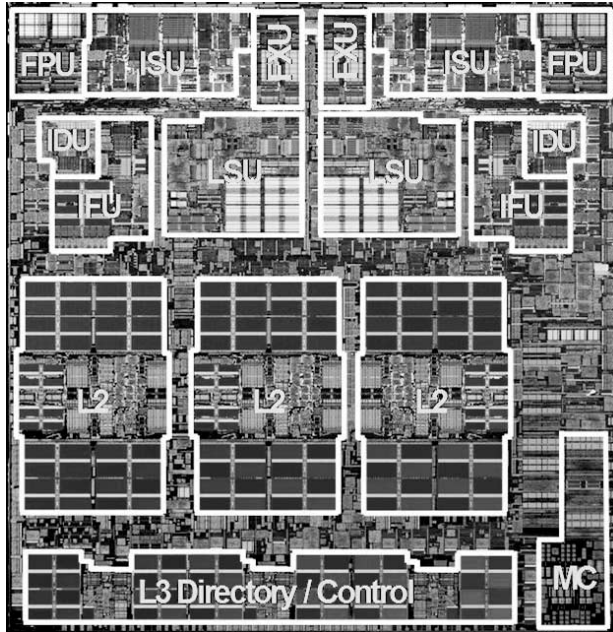
Figure 2. Power5 chip (FXU = fixed-point execution unit, ISU = instruction sequencing unit, IDU = instruction decode unit, LSU = load/store unit, IFU = instruction fetch unit, FPU = floating-point unit, and MC = memory controller).

supports a 1.875-Mbyte on-chip L2 cache. Power4 and Power4+ systems both have 32-Mbyte L3 caches, whereas Power5 systems have a 36-Mbyte L3 cache.

The L3 cache operates as a backdoor with separate buses for reads and writes that operate at half processor speed. In Power4 and Power4+ systems, the L3 was an inline cache for data retrieved from memory. Because of the higher transistor density of the Power5's 130-nm technology, we could move the memory controller on chip and eliminate a chip previously needed for the memory controller function. These two changes in the Power5 also have the significant side benefits of reducing latency to the L3 cache and main memory, as well as reducing the number of chips necessary to build a system.

Chip overview

Figure 2 shows the Power5 chip, which IBM fabricates using silicon-on-insulator (SOI) devices and copper interconnect. SOI technology reduces device capacitance to increase transistor performance.[5] Copper interconnect decreases wire resistance and reduces delays in wire-dominated chip-tim-

ing paths. In 130 nm lithography, the chip uses eight metal levels and measures 389 mm^2.

The Power5 processor supports the 64-bit PowerPC architecture. A single die contains two identical processor cores, each supporting two logical threads. This architecture makes the chip appear as a four-way symmetric multiprocessor to the operating system. The two cores share a 1.875-Mbyte (1,920-Kbyte) L2 cache. We implemented the L2 cache as three identical slices with separate controllers for each. The L2 slices are 10-way set-associative with 512 congruence classes of 128-byte lines. The data's real address determines which L2 slice the data is cached in. Either processor core can independently access each L2 controller.

We also integrated the directory for an off-chip 36-Mbyte L3 cache on the Power5 chip. Having the L3 cache directory on chip allows the processor to check the directory after an L2 miss without experiencing off-chip delays. To reduce memory latencies, we integrated the memory controller on the chip. This eliminates driver and receiver delays to an external controller.

Processor core

We designed the Power5 processor core to support both enhanced SMT and single-threaded (ST) operation modes. Figure 3 shows the Power5's instruction pipeline, which is identical to the Power4's. All pipeline latencies in the Power5, including the branch misprediction penalty and load-to-use latency with an L1 data cache hit, are the same as in the Power4. The identical pipeline structure lets optimizations designed for Power4-based systems perform equally well on Power5-based systems. Figure 4 shows the Power5's instruction flow diagram.

In SMT mode, the Power5 uses two separate instruction fetch address registers to store the program counters for the two threads. Instruction fetches (IF stage) alternate between the two threads. In ST mode, the Power5 uses only one program counter and can fetch instructions for that thread every cycle. It can fetch up to eight instructions from the instruction cache (IC stage) every cycle. The two threads share the instruction cache and the instruction translation facility. In a given cycle, all fetched instructions come from the same thread.
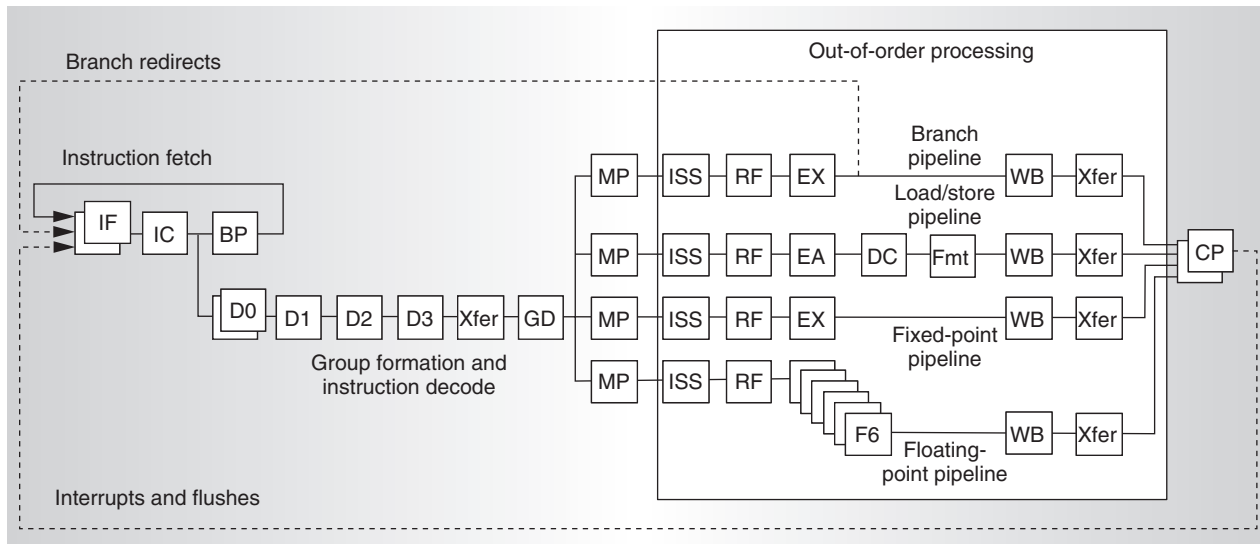
Figure 3. Power5 instruction pipeline (IF = instruction fetch, IC = instruction cache, BP = branch predict, D0 = decode stage 0, Xfer = transfer, GD = group dispatch, MP = mapping, ISS = instruction issue, RF = register file read, EX = execute, EA = compute address, DC = data caches, F6 = six-cycle floating-point execution pipe, Fmt = data format, WB = write back, and CP = group commit).
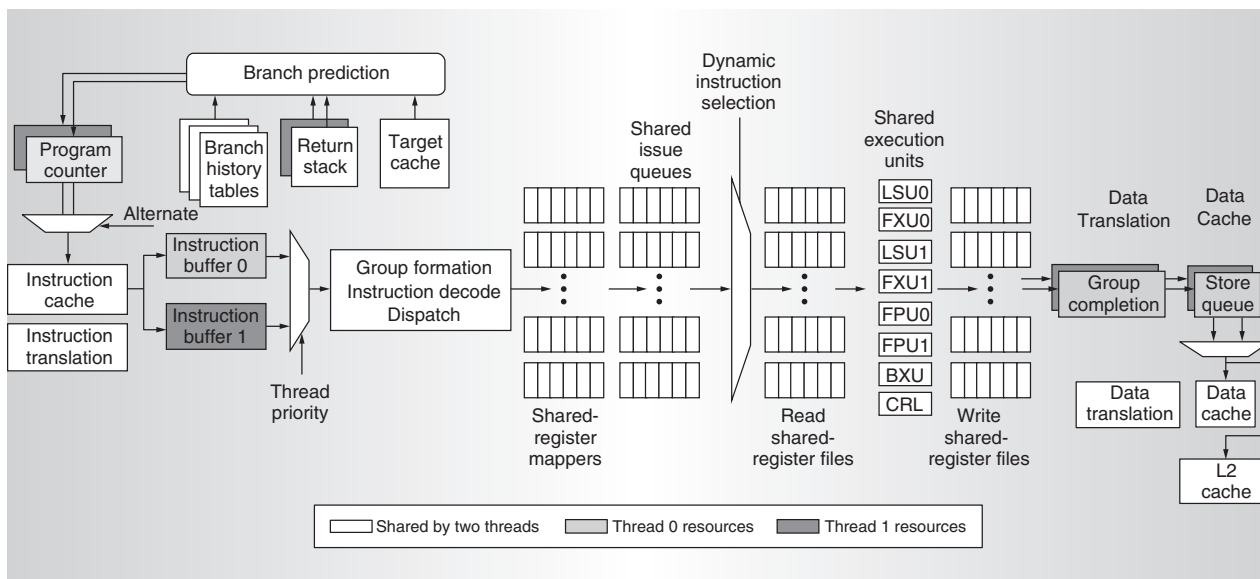

Figure 4. Power5 instruction data flow (BXU = branch execution unit and CRL = condition register logical execution unit).

The Power5 scans fetched instructions for branches (BP stage), and if it finds a branch, predicts the branch direction using three branch history tables shared by the two threads. Two of the BHTs use bimodal and path-correlated branch prediction mechanisms to predict branch directions.[6,7] The third BHT predicts which of these prediction mechanisms is more likely to predict the cor-rect direction.[7] If the fetched instructions contain multiple branches, the BP stage can predict all the branches at the same time. In addition to predicting direction, the Power5 also predicts the target of a taken branch in the current cycle's eight-instruction group. In the PowerPC architecture, the processor can calculate the target of most branches from the instruction's address and offset value. For

predicting the target of a subroutine return, the processor uses a return stack, one for each thread. For predicting the target of other branches, it uses a shared target cache. If there is a taken branch, the processor loads the program counter with the branch's target address. Otherwise, it loads the program counter with the address of the next sequential instruction to fetch from.

After fetching, the Power5 places instructions in the predicted path in separate instruction fetch queues for the two threads (D0 stage). Like the Power4, the Power5 can dispatch up to five instructions each cycle. On the basis of thread priorities, the processor selects instructions from one of the instruction fetch queues and forms a group (D1, D2, and D3 stages). All instructions in a group come from the same thread and are decoded in parallel.

Before a group can be dispatched, the processor must make several resources available for the instructions in the group. Each dispatched group needs an entry in the global completion table (GCT). Each instruction in the group needs an entry in an appropriate issue queue. Each load and store instruction needs an entry in the load reorder queue and store reorder queue, respectively, to detect out-of-order execution hazards.[1] When all the resources necessary for dispatch are available for the group, the group is dispatched (GD stage). Instructions flow through the pipeline stages between instruction fetch (IF) and group dispatch (GD) in program order. After dispatch, each instruction flows through the register-renaming (mapping) facilities (MP stage), which map the logical register numbers in the instruction to physical registers. In the Power5, there are 120 physical general-purpose registers (GPRs) and 120 physical floating-point registers (FPRs). The two threads dynamically share the register files. An out-of-order processor can exploit the high instruction-level parallelism exhibited by some applications (such as some technical applications) if a large pool of rename registers is available. To facilitate this, in ST mode, the Power5 makes all physical registers available to the single thread, allowing higher instruction-level parallelism.

After register renaming, instructions enter issue queues shared by the two threads. The Power5 microprocessor, like the Power4, has multiple issue queues: The floating-point issue queue feeds the two floating-point units, the branch issue queue feeds the branch execution unit, the condition register logical queue feeds the condition register logical operation execution unit, and a combined issue queue feeds the two fixed-point execution units and the two load-store execution units. Like the Power4, the Power5 contains eight execution units, each of which can execute an instruction each cycle.[1]

To simplify the logic for tracking instructions through the pipeline, the Power5 tracks instructions as a group. Each group of dispatched instructions takes an entry in the global completion table at the time of dispatch. The two threads share 20 entries in the GCT. Each GCT entry holds a group of instructions; a group can contain up to five instructions, all from the same thread. Power5 allocates GCT entries in program order for each thread at the time of dispatch. An entry is deallocated from the GCT when the group is committed. Although the entries in the GCT are in program order and from a given thread, successive entries can belong to different threads.

When all input operands for an instruction are available, it becomes eligible for issue. Among the eligible instructions in the issue queue, the issue logic selects one and issues it for execution (ISS stage). For instruction issue, there is no distinction between instructions from the two threads. When issued, the instruction reads its input physical registers (RF stage), executes on the proper execution unit (EX stage), and writes the result back to the output physical register (WB stage). Each floating-point unit has a six-cycle execution pipe (F1 through F6 stages). In each load-store unit, an adder computes the address to read or write (EA stage), and the data cache is accessed (DC stage). For load instructions, once data is returned, a formatter selects the correct bytes from the cache line (Fmt stage) and writes them to the register (WB stage).

When all the instructions in a group have executed (without generating an exception) and the group is the oldest group of a given thread, the group commits (CP stage). In the Power5, two groups can commit per cycle, one from each thread.

To efficiently support SMT, we tuned all resources for improved performance within

area and power budget constraints. The L1 instruction and data caches are the same size as in the Power4—64 Kbytes and 32 Kbytes—but their associativity has doubled to two- and four-way. The first-level data translation table is now fully associative, but the size remains at 128 entries.

Enhanced SMT features

To improve SMT performance for various workload mixes and provide robust quality of service, we added two features to the Power5 chip: dynamic resource balancing and adjustable thread priority.

Dynamic resource balancing. The objective of dynamic resource balancing is to ensure that the two threads executing on the same processor flow smoothly through the system. Dynamic resource-balancing logic monitors resources such as the GCT and the load miss queue to determine if one thread is hogging resources. For example, if one thread encounters multiple L2 cache load misses, dependent instructions can back up in the issue queues, preventing additional groups from dispatching and slowing down the other thread. To prevent this, resource-balancing logic detects that a thread has reached a threshold of L2 cache misses and throttles that thread. The other thread can then flow through the machine without encountering congestion from the stalled thread. The Power5 resource-balancing logic also monitors how many GCT entries each thread is using. If one thread starts to use too many GCT entries, the resource-balancing logic throttles it back to prevent its blocking the other thread.

Depending on the situation, the Power5 resource-balancing logic has three thread-throttling mechanisms:

- *Reducing the thread's priority* is the primary mechanism in situations where a thread uses more than a predetermined number of GCT entries.
- *Inhibiting the thread's instruction decoding until the congestion clears* is the primary mechanism for throttling a thread that incurs a prescribed number of L2 cache misses.
- *Flushing all the thread's instructions that are waiting for dispatch and holding the thread's decoding until the congestion clears* is the primary mechanism for throttling a thread executing a long-executing instruction, such as a synch instruction. (A synch instruction orders memory operations across multiple processors.)

Adjustable thread priority. Adjustable thread priority lets software determine when one thread should have a greater (or lesser) share of execution resources. (All software layers—operating systems, middleware, and applications—can set the thread priority. Some priority levels are reserved for setting by a privileged instruction only.) Reasons for choosing an imbalanced thread priority include the following:

- *A thread is in a spin loop waiting for a lock.* Software would give the thread lower priority, because it is not doing useful work while spinning.
- *A thread has no immediate work to do and is waiting in an idle loop.* Again, software would give this thread lower priority.
- *One application must run faster than another.* For example, software would give higher priority to real-time tasks over concurrently running background tasks.

The Power5 microprocessor supports eight software-controlled priority levels for each thread. Level 0 is in effect when a thread is not running. Levels 1 (the lowest) through 7 apply to running threads. The Power5 chip observes the difference in priority levels between the two threads and gives the one with higher priority additional decode cycles. Figure 5 (next page) shows how the difference in thread priority affects the relative performance of each thread. If both threads are at the lowest running priority (level 1), the microprocessor assumes that neither thread is doing meaningful work and throttles the decode rate to conserve power.

Single-threaded operation

Not all applications benefit from SMT. Having two threads executing on the same processor will not increase the performance of applications with execution-unit-limited performance or applications that consume all the chip's memory bandwidth. For this reason, the Power5 supports the ST execution
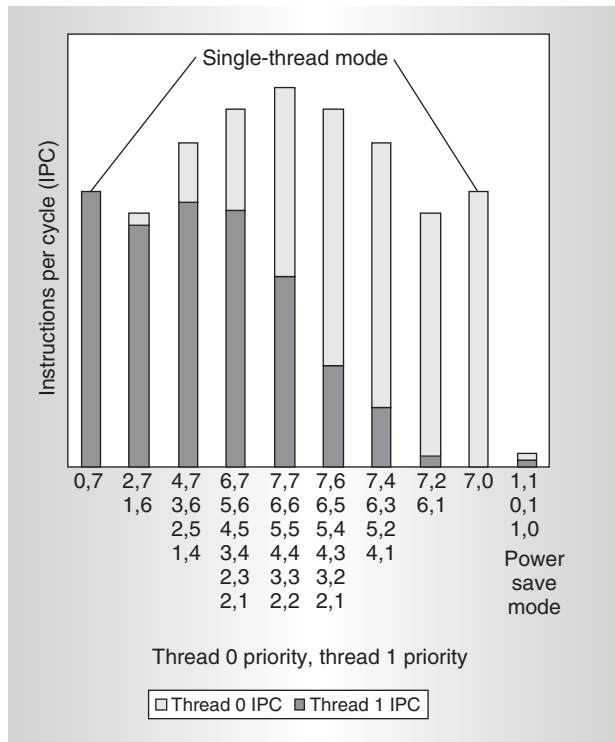
Figure 5. Effects of thread priority on performance.

mode. In this mode, the Power5 gives all the physical resources, including the GPR and FPR rename pools, to the active thread, allowing it to achieve higher performance than a Power4 system at equivalent frequencies.

The Power5 supports two types of ST operation: An inactive thread can be in either a dormant or a null state. From a hardware perspective, the only difference between these states is whether or not the thread awakens on an external or decrementer interrupt. In the dormant state, the operating system boots up in SMT mode but instructs the hardware to put the thread into the dormant state when there is no work for that thread. To make a dormant thread active, either the active thread executes a special instruction, or an external or decrementer interrupt targets the dormant thread. The hardware detects these scenarios and changes the dormant thread to the active state. It is software's responsibility to restore the architected state of a thread transitioning from the dormant to the active state.

When a thread is in the null state, the operating system is unaware of the thread's existence. As in the dormant state, the operating system does not allocate resources to a null thread. This mode is advantageous if all the system's executing tasks perform better in ST mode.

Dynamic power management

In current CMOS technologies, chip power has become one of the most important design parameters. With the introduction of SMT, more instructions execute per cycle per processor core, thus increasing the core's and the chip's total switching power. To reduce switching power, Power5 chips use a fine-grained, dynamic clock-gating mechanism extensively. This mechanism gates off clocks to a local clock buffer if dynamic power management logic knows the set of latches driven by the buffer will not be used in the next cycle. For example, if the GPRs are guaranteed not to be read in a given cycle, the clock-gating mechanism turns off the clocks to the GPR read ports. This allows substantial power saving with no performance impact.

In every cycle, the dynamic power management logic determines whether a local clock buffer that drives a set of latches can be clock gated in the next cycle. The set of latches driven by a clock-gated local clock buffer can still be read but cannot be written. We used power-modeling tools to estimate the utilization of various design macros and their associated switching power across a range of workloads. We then determined the benefit of clock gating for those macros, implementing cycle-by-cycle dynamic power management in macros where such management provided a reasonable power-saving benefit. We paid special attention to ensuring that clock gating causes no performance loss and that clock-gating logic does not create a critical timing path. A minimum amount of logic implements the clock-gating function.

In addition to switching power, leakage power has become a performance limiter. To reduce leakage power, the Power5 uses transistors with low threshold voltage only in critical paths, such as the FPR read path. We implemented the Power5 SRAM arrays mainly with high threshold voltage devices.

The Power5 also has a low-power mode, enabled when the system software instructs the hardware to execute both threads at the lowest available priority. In low-power mode, instructions dispatch once every 32 cycles at

most, further reducing switching power. The Power5 uses this mode only when there is no ready task to run on either thread.

The out-of-order execution Power5 design coupled with dual 2-way simultaneous multithreaded processors provides instruction and thread level parallelism. Future plans call for shrinking the size of the Power5 die by using a 90-nm lithography fabrication process, which should allow even higher performance at lower power.

Acknowledgments

We thank Ravi Arimilli, Steve Dodson, and the entire Power5 team.

References

1. J.M. Tendler et al., "Power4 System Micro-architecture," *IBM J. Research and Development*, vol. 46, no. 1, Jan. 2002, pp. 5-26.
2. J. Borkenhagen et al., "A Multithreaded Power PC Processor for Commercial Servers," *IBM J. Research and Development*, vol. 44, no. 6, Nov. 2000, pp. 885-898.
3. G. Alverson et al., "The Tera Computer System," *Proc. 1990 ACM Int'l Conf. Supercomputing* (Supercomputing 90), IEEE CS Press, 1990, pp. 1-6.
4. D.M. Tullsen, S.J. Eggers, and H.M. Levy, "Simultaneous Multithreading: Maximizing On-Chip Parallelism," *Proc. 22nd Ann. Int'l Symp. Computer Architecture* (ISCA 95), ACM Press, 1995, pp. 392-403.
5. G.G. Shahidi et al., "Partially-Depleted SOI Technology for Digital Logic," *Proc. Int'l Solid-State Circuits Conf.* (ISSCC 99), IEEE Press, 1999, pp. 426-427.
6. J.E. Smith, "A Study of Branch Prediction Strategies," *Proc. 8th Int'l Symp. Computer Architecture* (ISCA 81), IEEE CS Press, 1981, pp. 135-148.
7. S. McFarling, *Combining Branch Predictors*, tech. note TN-36, Digital Equipment Corp. Western Research Laboratory, 1993.

Ron Kalla is the lead engineer for IBM Power5 systems, specializing in processor core development. His research interests include computer micro-architecture and post-silicon hardware verification. Kalla has a BSEE in electrical engineering from the University of Minnesota.

Balaram Sinharoy is the chief scientist for the IBM Power5 microprocessor. His research interests include advanced microprocessor design, computer architecture, and performance analysis. Sinharoy has a BS in physics, a BTech in computer science and electrical engineering from the University of Calcutta, and an MS and a PhD in computer science from Rensselaer Polytechnic Institute. He is an IBM Master Inventor and a senior member of the IEEE.

Joel M. Tendler is the program director of technology assessment for the IBM Systems and Technology Group in Austin, Texas. His research interests include computer systems design, architecture, and performance. Tendler has a bachelor's degree in engineering from The Cooper Union, and a PhD in electrical engineering from Syracuse University. He is a member of the IEEE and the Computer Society.

Direct questions and comments about this article to Joel Tendler, IBM Corp., 0453B002, 11400 Burnett Road, Austin, TX 78758 or