IBM PowerPC 440 FPU with complex-arithmetic extensions

C. D. Wait

The PowerPC® 440 floating-point unit (FPU) with complex-arithmetic extensions is an embedded application-specific integrated circuit (ASIC) core designed to be used with the IBM PowerPC 440 processor core on the Blue Gene®/L compute chip. The FPU core implements the floating-point instruction set from the PowerPC Architecture™ and the floating-point instruction extensions created to aid in matrix and complex-arithmetic operations. The FPU instruction extensions define double-precision operations that are primarily single-instruction multiple-data (SIMD) and require two (primary and secondary) arithmetic pipelines and floating-point register files. However, to aid complex-arithmetic routines, some FPU extensions actually perform different (yet closely related) operations while executing in the arithmetic pipelines. The FPU core implements an operand crossbar between the primary and secondary arithmetic datapaths to enable each pipeline operand access from the primary or secondary register file. The PowerPC 440 processor core provides 128-bit storage buses and simultaneous issue of an arithmetic instruction with a storage instruction, allowing the FPU core to fully utilize the parallel arithmetic pipes.

Introduction

The IBM PowerPC* 440 (PPC440) floating-point unit (FPU) with complex-arithmetic extensions (PPC440 FP2) was the design point that resulted when we started with the original PPC440 FPU [1] and applied the Blue Gene*/L requirements of doubling FPU performance and improving cycle time, all on an aggressive schedule. The original PPC440 FPU [1] implemented a double-precision floating-point fused multiply–add pipeline and an independent load and store pipeline in IBM 0.18-μm 7SF technology. It attached to the PPC440A4 central processing unit (CPU) core using an auxiliary processor unit (APU) interface [2] and used the dual-issue ability of the CPU to keep the FPU arithmetic and storage pipelines utilized. The PPC440 FPU is PowerPC Book E [3] compliant and supports IEEE Standard 754 [4]. It was designed with an ASIC methodology (without its major functions, such as the multiplier or register file, being custom-implemented) to meet a cycle time of 525 MHz (at 1.8 V) in nominal silicon.

With this FPU as a starting point, we wanted to double the FPU throughput of Blue Gene/L and aid software workloads that make heavy use of double-precision operations in complex arithmetic and matrix multiplication. In order to achieve the requirements and meet the aggressive schedule, the PPC440 FP2 had to reuse as much as possible of the original FPU. Adding pipeline stages to allow doubling the frequency was considered too much of a redesign of the original FPU and would not have helped complex-arithmetic operations. Instead, it was decided to follow a single-instruction multiple-data (SIMD) approach and duplicate the original FPU arithmetic and storage pipelines. In addition to duplicating the pipelines, a second floating-point register (FPR) file was added. The result was two fused floating-point multiply–add (FMA) arithmetic

0018-8646/05/$5.00 © 2005 IBM

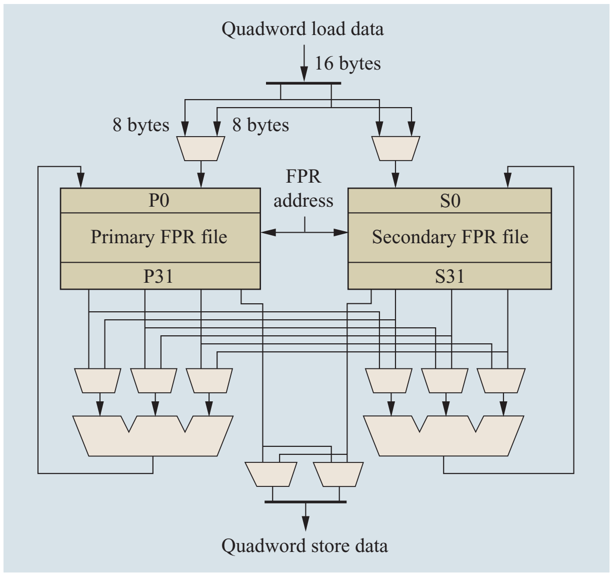

Figure 1

Architecture of the IBM PowerPC 440 FP2—primary and secondary data flow.

pipelines (primary and secondary), each fed by its own register file and load and store pipeline. The two fused multiply–add pipelines provide a peak throughput of two FMAs per cycle or four floating-point operations per cycle—twice the performance of the original PPC440 FPU core. To aid matrix and complex-arithmetic operations, the SIMD approach was modified to define different operations on the primary and secondary pipelines for a single instruction. To support this approach, an operand crossbar was implemented to allow either primary or secondary pipelines access to the data of the other.

Architecture

Figure 1 shows the primary and secondary arithmetic pipes, each with its own FPR file. The two register files share common addresses for each operand as specified by the instruction. If an instruction specifies the A operand to use address 3, both the primary and secondary register files will access address 3. Since the original PPC440 FPU did not have any special techniques (such as register renaming) to handle hazards or operand dependencies, the common addresses were important in allowing the reuse in the PPC440 FP2 of the complex hazard and dependency logic. The operands for the primary and secondary pipes could come from either pipe by means of the operand crossbar, but each pipe is allowed to write only to its respective register file. The ability to share operands between pipes enabled the creation of useful instructions that accelerate matrix and complex-arithmetic algorithms. A description of the algorithms that use the new floating-point instructions is given in [5].

Figure 1 also shows primary and secondary load and store datapaths, which use a quadword of storage to feed a doubleword to each pipe each cycle. Multiplexing allows the two doublewords to be swapped between the primary and secondary storage pipes before they read or write their respective register files using a common register address specified by the instruction. The primary load and store datapath supports the existing PowerPC floating-point storage instructions [3], while both pipes support new double-precision storage instruction extensions **(Table 1)** that access the primary and secondary register files.

The primary arithmetic pipe, along with its register file, executes the preexisting PowerPC floating-point instructions [3]. In addition, the primary pipe, along with the secondary pipe, implements the floating-point instruction extensions that take advantage of both pipes **(Table 2)**. The instructions include parallel FMAs, which

Table 1 Floating-point storage instruction extension examples.

Storage instruction	*Mnemonic*	*Description*	
		Primary	*Secondary*
Load floating parallel double indexed	lfpdx	$P_T = \mathrm{dw}(EA)$	$S_T = \mathrm{dw}(EA + 8)$
Load floating parallel single indexed	lfpsx	$P_T = \mathrm{w}(EA)$	$S_T = \mathrm{w}(EA + 4)$
Load floating secondary double indexed	lfsdx		$S_T = \mathrm{dw}(EA)$
Load floating cross double indexed	lfxdx	$P_T = \mathrm{dw}(EA + 8)$	$S_T = \mathrm{dw}(EA)$
Store floating parallel double indexed	stfpdx	$\mathrm{dw}(EA) = P_S$	$\mathrm{dw}(EA + 8) = S_S$
Store floating parallel single indexed	stfpsx	$\mathrm{dw}(EA) = P_S$	$\mathrm{dw}(EA + 4) = S_S$
Store floating secondary double indexed	stfsdx		$\mathrm{dw}(EA) = S_S$
Store floating cross double indexed	stfxdx	$\mathrm{dw}(EA + 8) = P_S$	$\mathrm{dw}(EA) = S_S$

Table 2 Floating-point instruction extension examples.

Instruction	*Mnemonic*	*Description*	
		Primary	*Secondary*
Floating parallel multiply–add	fpmadd	$P_T = P_A \cdot P_C + P_B$	$S_T = S_A \cdot S_C + S_B$
Floating parallel negative multiply subtract	fpnmsub	$P_T = -P_A \cdot P_C + P_B$	$S_T = -S_A \cdot S_C + S_B$
Floating cross multiply–add	fxmadd	$P_T = S_A \cdot P_C + P_B$	$S_T = P_A \cdot S_C + S_B$
Floating cross copy–primary multiply–add	fxcpmadd	$P_T = P_A \cdot P_C + P_B$	$S_T = P_A \cdot S_C + S_B$
Floating cross copy–secondary multiply–add	fxcsmadd	$P_T = S_A \cdot P_C + P_B$	$S_T = S_A \cdot S_C + S_B$
Asymmetrical cross copy–primary nsub–primary multiply–add	fxcpnpma	$P_T = -P_A \cdot P_C + P_B$	$S_T = P_A \cdot S_C + S_B$
Asymmetrical cross copy–secondary nsub–secondary multiply–add	fxcsnsma	$P_T = S_A \cdot P_C + P_B$	$S_T = -S_A \cdot S_C + S_B$
Floating cross cmplx nsub–primary multiply–add	fxcxnpma	$P_T = -S_A \cdot S_C + P_B$	$S_T = S_A \cdot P_C + S_B$
Floating cross cmplx nsub–secondary multiply–add	fxcxnsma	$P_T = S_A \cdot S_C + P_B$	$S_T = -S_A \cdot P_C + S_B$
Floating parallel reciprocal estimate	fpre	$P_T = \text{RecipEst}(P_B)$	$S_T = \text{RecipEst}(S_B)$
Floating parallel select	fpsel	$P_T = P_A \, ? \, P_C : P_B$	$S_T = S_A \, ? \, S_C : S_B$
Floating secondary compare[1]	fscmp		$\text{CR}[BF] = S_A <> S_B$
Floating cross move	fxmr	$P_T = S_B$	$S_T = P_B$

execute the same operation in each pipe with their respective pipe operands; cross FMAs, which execute the same operation in each pipe but with operands from the opposite pipe; and asymmetric and complex FMAs, which execute different FMA-type instructions with operands from either pipe.

With the operand crossbar, many combinations would be possible, including single-precision versions of the instructions. However, since each combination would require a new opcode, and opcode space was limited, only double-precision versions of the instructions thought to be the most useful for complex-arithmetic and matrix operations were provided. Move-type instructions, along with other parallel non-FMA instructions, were included to finish the new instruction set. Some of these are parallel instructions for generating reciprocal estimates, reciprocal square-root estimates, and even a parallel floating-point select instruction. The newly defined operations executing in the primary pipe and secondary pipe behave the same as their PowerPC floating-point instruction equivalent [3], with the exception of the Floating-Point Status and Control Register (FPSCR), as described next.

FPSCR

The floating-point instruction extensions treat exceptions as disabled and do not update the PowerPC FPSCR. However, the instructions do follow the rounding mode and non-IEEE bit control fields. It was thought that the default handling of exceptions, as defined by IEEE Standard 754 [4], would be sufficient for the target applications. The complexity of handling enabled exceptions and updating the FPSCR for the SIMD-like instructions, especially when the SIMD-like instructions are mixed with PowerPC floating-point instructions, would outweigh the benefits. This makes the PPC440 FP2 consistent with the IEEE Standard 754, but not strictly compliant.

Implementation

The PPC440 FP2 core was designed in IBM 0.13-μm 8SF technology to be attached to the dual-issue PPC440G5 CPU [2] processor core using the APU interface. A beneficial aspect of the APU interface is its ability to allow the "auxiliary processor" to implement new instructions without changes to the PPC440G5 CPU core. The CPU provides the instruction stream across the APU interface, allowing the APU to decode instructions it recognizes. The PPC440 FP2 core was able to implement all of the newly defined instructions without any impact to the CPU core, resulting in significant schedule and design effort savings.

Since a storage instruction using the APU interface with its quadword storage path can provide two double-precision operands per cycle, both primary and secondary arithmetic pipes can execute double-precision FMAs

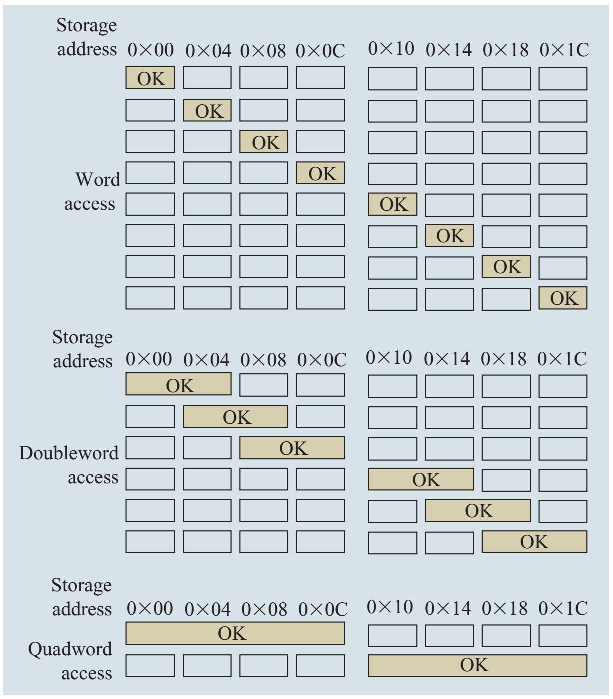
Storage address
0×00 0×04 0×08 0×0C 0×10 0×14 0×18 0×1C
OK
Word access
Storage address
0×00 0×04 0×08 0×0C 0×10 0×14 0×18 0×1C
OK
Doubleword access
Storage address
0×00 0×04 0×08 0×0C 0×10 0×14 0×18 0×1C
OK
Quadword access

Figure 2

Alignment restrictions.

without becoming starved for data. For the PPC440G5 CPU to be able to efficiently provide a doubleword or quadword of storage each cycle across the APU interface, data alignment restrictions must be followed. Storage addresses must start on a word boundary, and the access cannot cross a quadword (16-byte) boundary **(Figure 2)**. The cache architecture of the CPU has a 32-byte cache line and allows only 16 bytes to come from one half of the cache or the other. Misaligned accesses cause an alignment interrupt, creating a significant performance penalty.

The new SIMD-like instructions execute in the parallel PPC440 FP2 arithmetic pipes the same way as the similar PowerPC floating-point instructions [3] (but with exceptions disabled in the FPSCR). When a parallel FMA is issued across the APU interface by the CPU, the FP2 splits the instruction into separate FMA operations, as defined by the opcode, and selects the appropriate operands from the crossbar logic. Since the primary and secondary arithmetic pipes are basically copies of each other, the two pipes execute the floating multiply–add operations synchronously as they travel down the pipelines. Typical FMA latency is five cycles [1], with each pipe updating its own register file without causing any changes to the FPSCR.

Operand hazards

One of the advantages of the new SIMD-like instructions was the ability of the PPC440 FP2 core to reuse the hazard and dependency logic of the original FPU. Reuse of this logic reduced the design effort and improved quality by taking advantage of previous verification efforts on this complex logic in the original FPU. However, as is sometimes the case, simple solutions have surprise problems.

As seen in **Figure 3**, each arithmetic pipe contains operand bypass datapaths around the register files—one bypass from the arithmetic result and one bypass from the load pipe. A typical read-after-write (RAW) hazard, in which a load target is bypassed to an FMA, would be the following:[1]

Instruction	Description	FP2 implementation
lfd	F3 $\leftarrow$ dw(EA)	$\boldsymbol{P_3} \leftarrow \mathrm{dw}(EA)$
fmadd	F4 $\leftarrow$ F3, F2, F1	$P_4 \leftarrow \boldsymbol{P_3} \cdot P_2 + P_1$

P_3 represents F3 in the primary pipe.

The same hazard logic works equally as well for a parallel load and parallel FMA:

Instruction	Description	FP2 implementation
lfpdx	F3 $\leftarrow$ qw(EA)	$\boldsymbol{P_3} \leftarrow \mathrm{dw}(EA)$
		$\boldsymbol{S_3} \leftarrow \mathrm{dw}(EA + 8)$
fpmadd	F4 $\leftarrow$ F3, F2, F1	$P_4 \leftarrow \boldsymbol{P_3} \cdot P_2 + P_1$
		$S_4 \leftarrow \boldsymbol{S_3} \cdot S_2 + S_1$

P_3 represents F3 in the primary pipe; S_3 represents F3 in the secondary pipe.

Both the primary and secondary load targets (F3) are bypassed to their respective primary and secondary arithmetic pipes. Difficulties begin to appear when the instruction stream contains a mix of parallel and nonparallel instructions. Nonparallel instructions have only one register file as a target, such as existing PowerPC floating-point instructions (arithmetic or loads that target only the primary register file) or new load instructions that target only the secondary register file:

Instruction	Description	FP2 implementation
lfd	F3 $\leftarrow$ dw(EA)	$\boldsymbol{P_3} \leftarrow \mathrm{dw}(EA)$
fpmadd	F4 $\leftarrow$ F3, F2, F1	$P_4 \leftarrow \boldsymbol{P_3} \cdot P_2 + P_1$
		$S_4 \leftarrow \underline{S_3} \cdot S_2 + S_1$

The hazard detection logic sees the RAW hazard for F3 and naturally seeks to enable the bypass datapath for both pipes. However, the hazard really exists only for the primary pipe; a false hazard exists for the secondary pipe. F3 for the secondary pipe should come from the register

[1]Boldface and underscores represent FPR dependencies or hazards between instructions. For example, P_3 indicates primary FPR 3, which is the target of one instruction and is then used as a source in the next instruction. Because of the characteristics of the new FP instructions, sometimes a dependency or hazard may not be real. If there is a real hazard, the term is shown in boldface. If the hazard is not real, the term is underscored.

file, not from the bypass datapath. An example of another problem that had to be solved is the following:

Instruction	Description	FP2 implementation
lfd	F3 $\leftarrow$ dw(EA)	$\mathbf{P_3} \leftarrow \mathrm{dw}(EA_1)$
lfsdx	F3 $\leftarrow$ dw(EA)	$\underline{S}_3 \leftarrow \mathrm{dw}(EA_2)$
fpmadd	F4 $\leftarrow$ F3, F2, F1	$P_4 \leftarrow \mathbf{P_3} \cdot P_2 + P_1$
		$S_4 \leftarrow \underline{S}_3 \cdot S_2 + S_1$

This example contains only a secondary pipe load floating-point double (lfsdx) along with a PowerPC lfd instruction. These instructions, combined with a parallel FMA, create an apparent write-after-write (WAW) hazard and a RAW hazard. The WAW hazard does not really exist, since the two load commands are actually writing to different register files; both writes to F3 must be allowed to complete. The RAW hazard appears to the logic as one dependency with the most recent load (lfsdx), but in fact there are two dependencies. The PPC440 FP2 allows both load instructions to bypass data to the parallel FMA.

Design reuse

The original PPC440 FPU was partitioned into the following units: instruction decode and issue (IDI), register and pipe management (RPM), load and store control (LSC), and the arithmetic status and exceptions (ASE). The IDI maintained its entire logical structure in the FP2 core, with the only changes coming in the additional decodes of the new instructions. As described above, the original RPM, which included the complex hazard logic, needed refining, but it was still largely reused for the PPC440 FP2.

The LSC for the FP2 core duplicated the double-precision data flow of the original LSC into a primary and secondary pipe data flow. Additional logic was added to provide multiplexing of quadword load data into doublewords for both pipes and merging doubleword store data into quadwords for stores.

The ASE, which includes the arithmetic data pipe, made up approximately 65% of the original FPU core and was entirely duplicated for the secondary pipe. After duplication, the sections dealing with the FPSCR and floating-point divide were removed from the secondary pipe to save area and power. The floorplan of the PPC440 FP2 showing each unit can be seen in **Figure 4**.

The physical implementation of each unit followed an ASIC methodology in that most of the design was synthesized into gates from the IBM 8SF technology library. No custom complementary metal oxide semiconductor (CMOS) logic macros were used, but some key functions were implemented with manually placed specialized ASIC gates to optimize timing. For example, the multiplier made use of 4:2 compressor gates to provide an "add" function for the multiplier partial product bit vectors. The 4:2 compressor was implemented as an ASIC circuit and treated as an extension to the 8SF library.

In addition to the multiplier timing challenge, another problem involved implementing the operand crossbar to meet the cycle time. As can be seen in Figure 3, the solution chosen implements the crossbar in the first

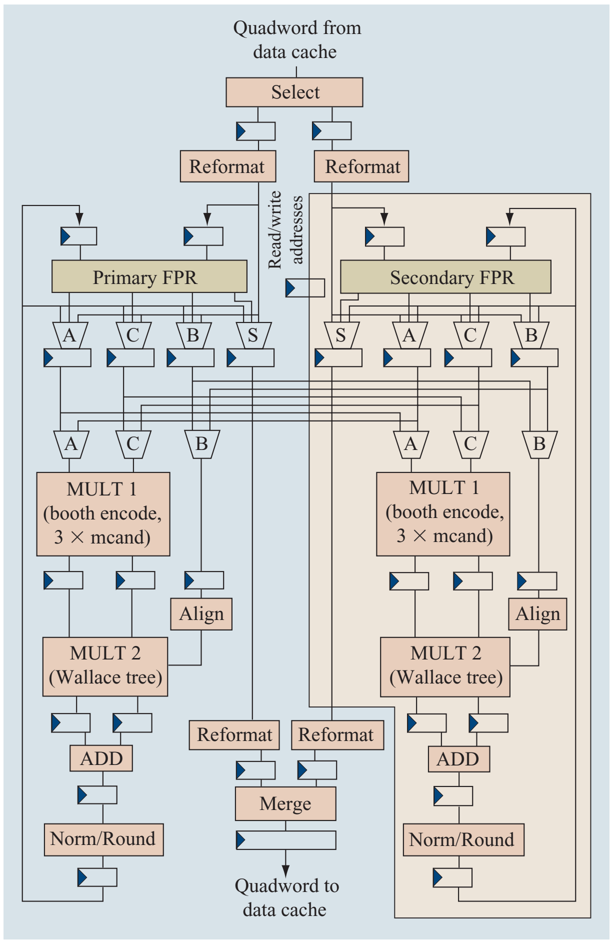

Figure 3

PPC440 FP2 data flow.

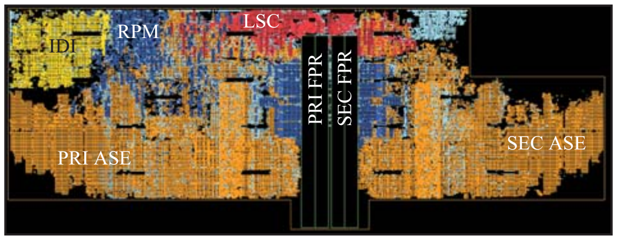

Figure 4

PPC440 FP2 floorplan.

pipeline stage of the multiplier. The basic reason was that the first stage had the most cycle time flexibility because of the multiplier design. On the floorplan, the primary and secondary register files are located side by side with their arithmetic pipes extending in opposite directions. The first stage begins near the register files, with the multiplier employing a radix-8 Booth encode. This choice of encode requires calculating ($3 \times$ multiplicand), which was the longest delay path of the first stage. With careful gate placement and wiring, the additional delay of operands crossing over the register files from the opposite pipe through the crossbar selectors and the $3 \times$ multiplicand calculation was all accomplished in one cycle.

Conclusion

The PPC440 FP2 with complex-arithmetic extensions was accomplished with an ASIC design methodology on an aggressive schedule. A great deal of effort went into placement and wiring to achieve running at 800 MHz (nominal process) in IBM 0.13-μm 8SF technology. The reuse of major functions from the original FPU and rigorous simulation combined to make the PPC440 FP2 functionally error-free with the first-pass design.

Acknowledgments

The author would like to acknowledge the major contribution of Kenneth Dockser for his work on the initial design and architecture used by the PPC440 FP2. The Blue Gene/L project has been supported and partially funded by the Lawrence Livermore National Laboratory on behalf of the United States Department of Energy under Lawrence Livermore National Laboratory Subcontract No. B517552.

*Trademark or registered trademark of International Business Machines Corporation.

References

1. K. Dockser, "'Honey, I Shrunk the Supercomputer,'—The PowerPC 440 FPU Brings Supercomputing to IBM's Blue Logic Library," *IBM MicroNews* **7**, No. 4, 27–29 (Fourth Quarter 2001); see *http://www-306.ibm.com/chips/micronews/vol7_no4/mn_vol7_no4_fnl.pdf*.
2. IBM Corporation, PPC440x5 CPU Core User's Manual, Publication No. SA14-2613-03, June 2001; see *http://www-306.ibm.com/chips/techlib/techlib.nsf/products/PowerPC_440_Embedded_Core*.
3. IBM Corporation, Book E: Enhanced PowerPC Architecture, March 2000.
4. *IEEE Standard 754*, "IEEE Standard for Binary Floating-Point Arithmetic," ©1985 IEEE.
5. S. Chatterjee, L. R. Bachega, P. Bergner, K. A. Dockser, J. A. Gunnels, M. Gupta, F. G. Gustavson, C. A. Lapkowski, G. K. Liu, M. Mendell, R. Nair, C. D. Wait, T. J. C. Ward, and P. Wu, "Design and Exploitation of a High-Performance SIMD Floating-Point Unit for Blue Gene/L," *IBM J. Res. & Dev.* **49**, No. 2/3, 377–391 (2005, this issue).

Received May 6, 2004; accepted for publication August 23, 2004; Internet publication March 23, 2005

Charles D. Wait *IBM Engineering and Technology Services, 3605 Highway 52 N., Rochester, Minnesota 55901 (cdwait@us.ibm.com).* Mr. Wait was the PPC440 FP2 team leader. He began his career with IBM in Kingston, New York, in 1986 after graduating from Iowa State University with a B.S. degree in computer engineering. He worked on multiple hardware implementations of the System/390* vector processor in a variety of development positions from logic entry to floating-point team leader. He transferred to IBM Rochester in 1993 to work on system integration simulation of the central electronics complex for the IBM AS/400*. He later worked in floating-point hardware development and as processor timing leader of the star processors used in the IBM iSeries* and pSeries* systems. Mr. Wait has received numerous formal and informal awards, including an IBM Outstanding Technical Achievement Award for his work as the star processor timing leader.