To appear in *25 years of the International Symposia of Computer Architecture: Selected Papers*, ed. Gurindar Sohi

Reflections on "Tempest and Typhoon: User-Level Shared Memory"

Steven K. Reinhardt,* James R. Larus, and David A. Wood

Computer Sciences Department
University of Wisconsin–Madison
1210 West Dayton Street
Madison, WI 53706
wwt@cs.wisc.edu

Introduction

Tempest and Typhoon have emerged as among the most influential contributions of the Wisconsin Wind Tunnel project, a collaborative effort with Prof. Mark D. Hill, several staff members, and a large group of graduate students. This retrospective focuses on the origins of the Tempest and Typhoon ideas and their subsequent evolution.

The Beginnings

The seeds of the project began to germinate in late 1990 and early 1991 with our effort to rapidly prototype large-scale shared-memory multiprocessors. Because other research groups had a one- to two-year lead in their prototyping efforts—and considerably more resources—our project started with the goal of exploiting the parallel computers that our department was acquiring with funding from NSF's Institutional Infrastructure program.

During this exploratory phase, we made the essential observation that shared-memory systems permit a continuum of implementations, ranging from full hardware support to software simulation/emulation on a message-passing platform. Moreover, in the middle lies a rich collection of mixed hardware/software design alternatives.

An internal research note, dated July 9, 1991, roughly classified these alternatives into five levels:

Level 0: Software simulation/emulation. At this level, shared-memory programs execute on an unmodified message-passing parallel platform. A program's loads and stores are replaced with calls to routines that simulate the shared-memory behavior of the proposed design.

Level 1: Shared virtual memory. This level incorporates Kai Li's observation that address translation hardware can be used to map shared memory references to local pages and detect non-local references, albeit at coarse granularity.

Level 2: Fine-grain shared virtual memory. This level makes the observation that shared virtual memory can be implemented at a finer granularity given a mechanism—such as fine-grain "presence" bits—to detect when cache blocks are not stored locally.

Level 3: Local hardware support. This level begins to blur the distinction between a test-bed and a prototype. It extends level 2 with hardware support to initiate requests and handle responses on misses to remote data.

Level 4: Remote hardware support. The final level adds hardware support to handle external requests to a node's memory—that is, a directory controller. This last level encompasses all-hardware implementations.

Initially, we considered these approaches solely as alternatives for evaluating the hardware of interest, a highly integrated hardware-centric system. This discussion lead to the development of the Wisconsin Wind Tunnel (WWT), the parallel simulation system that gave our project its name [9]. The original version of WWT used a parallel message passing machine (a Thinking Machines CM-5) to simulate a hypothetical shared memory machine. WWT is a hybrid of levels 0 and 2, and uses the CM-5's ECC bits to implement fine-grain valid bits. Memory references that access non-local shared memory cause a trap, because of either a page fault or an intentionally set ECC error. Fine-grain access control allowed direct execution of shared-memory programs, which resulted in a very fast simulator that permitted rapid evaluation of hypothetical shared-memory implementations.

Cooperative Shared Memory

WWT was originally developed to evaluate an architectural approach called Cooperative Shared Memory (CSM) [4]. CSM's central premise was that hardware and software could cooperate to support shared memory efficiently. This cooperation took two forms. First, a programming performance model helped programmers identify expensive operations (so they could avoid them when possible) and helped hardware designers identify common

* Currently at The University of Michigan, EECS Department, 1301 Beal Ave., Ann Arbor, MI 48109-2122; email: stever@eecs.umich.edu.

cases (so they could optimize them). Second, CSM encouraged hardware designers to concentrate expensive hardware resources on optimizing frequent operations and to fall back to software for complex, less frequent cases.

Our programming performance model was called Check-In/Check-Out (CICO). It asked programmers to issue an advisory `check_out` directive before the expected first use of shared data followed by a `check_in` directive after the expected last use. We further proposed Dir_1SW, a minimal directory protocol that supported CICO-conforming programs efficiently (i.e., entirely in hardware). Violations of the CICO model, which often required more complex protocol operations, were handled correctly but less quickly by trapping to software. A later version of Dir_1SW, called Dir_1SW+, handled some common CICO violations in hardware as well [12].

Cooperative Shared Memory provided the philosophic underpinnings of Tempest and Typhoon. Hardware and software should cooperate to achieve good shared-memory performance. Programmers should be able to optimize performance by exploiting hardware mechanisms. Hardware designers should focus on providing efficient hardware mechanisms, and, as much as possible, leave policy to software.

WWT as a Shared Memory Machine

While designing and developing the Wisconsin Wind Tunnel, we met developers of the emerging generation of MPPs, the Intel Paragon and Thinking Machines CM-5. During these meetings, a frequent misconception was that WWT was a "real" shared-memory system, not just a test-bed. Students running programs on WWT also tended to blur this distinction.

In early 1993, we recognized that WWT was an interesting fine-grain shared-memory system in its own right, an observation that led to two parallel efforts. First, we began to develop a performance-oriented shared-memory system for the CM-5, simply by removing from WWT the components that calculated the performance of the hypothetical hardware. This effort led to the Blizzard system (discussed further below).

Second, we realized that a small amount of hardware support might allow a message passing machine to achieve competitive shared memory performance. Our first step in this direction was a joint project with Thinking Machines and NimBus to develop an enhanced memory controller (EMC) that provided first-class fine-grain access control. The short-term goal was to eliminate the complex, relatively slow "hacks" required by WWT to manipulate ECC and to synthesize a fine-grain read-only state via page protection. The longer term goal was to develop a "smart NI" that could handle the most frequent cases of a simple Dir_1SW-like coherence protocol—most likely with a programmable processor. The EMC chip was designed and fabricated by NimBus. Sadly, Thinking Machines never used it in a product, largely because of the additional product risk posed by the enhanced features.

Typhoon

Typhoon emerged as the follow-on to the EMC and "smart NI" approach. To minimize our exposure to Thinking Machines's marketing decisions, we envisioned a single ASIC that would not interfere with "normal" operations within a local node. The ASIC would provide hardware snooping support for fine-grain access control, an embedded protocol processor to implement some or all of the coherence protocol, and a closely coupled network interface.

A major goal of Typhoon was to increase programming flexibility beyond CSM, allowing programmers to optimize known communication patterns aggressively. The approach that we chose was to give programmers direct access to the raw mechanisms underlying shared-memory protocols. An important difference between Typhoon and our earlier Dir_1SW work came from our realization that many protocols we envisioned needed flexibility on the requester side, not just on the directory side. This approach fit well with the "smart NI" model that called for using a programmable processor or controller to access the network interface. We refer the reader back to the original paper for the rest of the motivation and design.

Tempest

Programmers needed an abstraction of Typhoon's shared-memory mechanisms to develop protocols. Initially, we borrowed from the internal WWT interfaces and assigned each memory block an access control tag. Accesses that conflicted with the referenced block's tag invoked a user-specified handler. We initially referred to this abstraction simply as the "tagged block model".

Two important changes occurred in late 1993. First, we recognized the fundamental importance of the programming abstraction. The tagged block model applied equally well to the nascent all-software Blizzard system as to Typhoon, and it clearly made sense to support the same protocol programming interface on both systems. Although our original intent was merely to develop a simple abstraction for Typhoon, we ended up with a powerful abstraction for which Typhoon was just one implementation. Second, we gave the abstraction a "first class" name to reflect our appreciation for its importance. We chose Tempest, in keeping with the

Wind Tunnel group's practice of naming nearly everything after a wind (fortunately, children have been unaffected by this practice).[1]

Subsequently, the Tempest interface [5] became the focus of much of the WWT group's research. Tempest's stable, powerful abstractions enabled parallel, synergistic research on both sides of the interface. On the system side, we began to explore the broad range of possible Tempest implementations. Other group members simultaneously investigated the implications of a flexible protocol interface for applications, programmers, and compilers. A key goal emerged to have Tempest provide application portability across a diverse range of implementations, each with different cost/performance objectives.

Blizzard: An All-software Tempest System

The Blizzard systems are a family of Tempest implementations that run on stock hardware [11]. One variant, Blizzard-E, uses WWT's "ECC hack" to provide fine-grain access control. Another variant, Blizzard-S, uses executable editing [7] to add explicit inline checks. Both versions were initially implemented on the CM-5 and later ported to the Wisconsin COW, our cluster of 40 Myrinet-connected Sun SparcStation 20s. Our research on the applications of Tempest benefited greatly from the availability of a real (not simulated), relatively stable Tempest platform.

Typhoon-0: Minimal Hardware for Tempest

A key aspect of the Typhoon design is the (ab)use of existing snooping cache coherence protocols to provide hardware fine-grain access control on an otherwise unmodified platform. We decided to demonstrate the feasibility of this approach by implementing a prototype access control board for the Sun SparcStation and populating the 40 nodes of the COW. The resulting system, Typhoon-0 [10], can be viewed either as a prototype of the more highly integrated Typhoon or as a minimal-hardware Tempest implementation. Unlike Typhoon, Typhoon-0 relies on off-the-shelf devices for each node's network interface and protocol processor. In the process of our design and analysis of Typhoon-0, we recognized the benefits of an intermediate design, Typhoon-1 [10], that integrates the network interface (but not the protocol processor) with Typhoon-0's access control unit.

1. This practice created an unnecessary amount of confusion among the meteorologically challenged, who could not tell a Tempest from a Typhoon.

Custom Protocol Demonstrations

One of our early experiments investigated the performance gains made possible by writing custom, application-specific protocols [3]. The performance improvements for three application kernels on the 32-node Blizzard/CM-5 system ranged from 1.4–16 times, which strongly encouraged us to extend this approach. Subsequent experiments [8] also demonstrated the value of custom protocols in running parallel irregular applications. However, the efforts of many students showed that writing custom protocols was difficult and time consuming.

Programming Support

In response to these problems, the project investigated programming languages and tools to support custom protocol development. One effort lead to the Teapot language for writing and verifying custom protocols [2]. This language halved the size of a protocol, but more importantly, enabled use of automatic verification tools, drastically reducing the time and effort to produce a working protocol.

Another attack on the difficulty of writing protocols, was to shift the burden of exploiting them from a programmer to a compiler. Several efforts clearly showed that compilers for high-level programming languages could exploit custom protocols, to produce code with robust parallel performance that in many cases exceeded hand-written code. Initially, this work focused on research parallel languages, such as C**, in which Tempest supported a novel parallel programming model [6]. However, with the assistance of the Portland Group, we were also able to show that custom protocols could greatly expand the range of High Performance Fortran (HPF) programs that ran well [1].

Summary

The Tempest and Typhoon paper was the first of a broad collection of Tempest-related papers from the Wisconsin Wind Tunnel project (see `http://www.cs.wisc.edu/~wwt`). Its impact within Wisconsin has been considerable, contributing to 8 Ph.D. dissertations and 8 Masters degrees. We suspect its impact beyond Wisconsin has also been considerable, but we leave that evaluation to others.

Acknowledgments

Many have asked about the absence of Mark Hill as an author of the Typhoon papers. Mark has been a constant co-leader and contributor to the Wind Tunnel project, including the Tempest and Typhoon research. In mid-summer 1993, Mark unilaterally distanced himself from the Typhoon effort—against our objections—

to help potential tenure letter writers differentiate our contributions from his own. Despite staying at arm's length, Mark made numerous contributions to this work and has been actively involved in the Tempest follow-on projects.

Many students have contributed to the Tempest work. We would like to single out one, Rob Pfile, for his efforts to implement the Typhoon-0 prototype.

This work received financial support from a number of sources. Initial support came from the National Science Foundations PYI/NYI program (grants CCR-9157366 and CCR-9357779). Primary support for Typhoon came from Michael Foster of the NSF's Experimental Systems program (grant MIP-9225097). Gil Weigand and Bob Lucas of the Defense Advanced Research Projects Agency supported the Blizzard implementations (ARPA Order Number B550). Dave Douglas, Bob Zak, and Greg Papadopolous provided technical and financial support from Thinking Machines Corporation and later Sun Microsystems. Additional support was provided by a Univ. of Wisconsin Graduate School Grant, a Wisconsin Alumni Research Foundation Fellowship, an AT&T Ph.D. Fellowship, and donations from Digital Equipment Corporation, Xerox Corporation, the Portland Group. Our Thinking Machines CM-5 and Wisconsin COW were purchased through NSF Institutional Infrastructure Grant CDA-9024618 with matching funding from the Univ. of Wisconsin Graduate School.

About the Authors:

Steven K. Reinhardt completed his PhD at the University of Wisconsin on the Typhoon implementations of the Tempest interface. He is currently an Assistant Professor of Electrical Engineering and Computer Science at the University of Michigan, where he is conducting research on parallel computer architectures and systems.

James R. Larus is an Associate Professor of Computer Sciences at the University of Wisconsin–Madison. His research includes programming languages and compilers, the design and programming of shared-memory parallel computers, program profiling and tracing, and program executable editing.

David A. Wood is an Associate Professor of Computer Sciences and Electrical and Computer Engineering at the University of Wisconsin–Madison. His research spans computer architecture, emphasizing parallel computer design, implementation, and evaluation.

References

[1] S. Chandra and J. R. Larus. Optimizing communication in HPF programs on fine-grain distributed shared memory. In *Sixth ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, pages 100–111, June 1997.

[2] S. Chandra, B. Richards, and J. R. Larus. Teapot: Language support for writing memory coherence protocols. In *Proceedings of the SIGPLAN '96 Conference on Programming Language Design and Implementation (PLDI)*, May 1996.

[3] B. Falsafi, A. R. Lebeck, S. K. Reinhardt, I. Schoinas, M. D. Hill, J. R. Larus, A. Rogers, and D. A. Wood. Application-specific protocols for user-level shared memory. In *Proceedings of Supercomputing '94*, pages 380–389, Nov. 1994.

[4] M. D. Hill, J. R. Larus, S. K. Reinhardt, and D. A. Wood. Cooperative shared memory: Software and hardware for scalable multiprocessors. *ACM Transactions on Computer Systems*, 11(4):300–318, Nov. 1993. Earlier version appeared in ASPLOS V.

[5] M. D. Hill, J. R. Larus, and D. A. Wood. Tempest: A substrate for portable parallel programs. In *Proceedings of COMPCON '95*, pages 327–332, San Francisco, California, Mar. 1995.

[6] J. R. Larus, B. Richards, and G. Viswanathan. LCM: Memory system support for parallel language implementation. In *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VI)*, pages 208–218, Oct. 1994.

[7] J. R. Larus and E. Schnarr. EEL: Machine-independent executable editing. In *Proceedings of the SIGPLAN '95 Conference on Programming Language Design and Implementation (PLDI)*, pages 291–300, June 1995.

[8] S. S. Mukherjee, S. D. Sharma, M. D. Hill, J. R. Larus, A. Rogers, and J. Saltz. Efficient support for irregular applications on distributed-memory machines. In *Fifth ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming (PPOPP)*, July 1995.

[9] S. K. Reinhardt, M. D. Hill, J. R. Larus, A. R. Lebeck, J. C. Lewis, and D. A. Wood. The Wisconsin Wind Tunnel: Virtual prototyping of parallel computers. In *Proceedings of the 1993 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 48–60, May 1993.

[10] S. K. Reinhardt, R. W. Pfile, and D. A. Wood. Decoupled hardware support for distributed shared memory. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 34–43, May 1996.

[11] I. Schoinas, B. Falsafi, A. R. Lebeck, S. K. Reinhardt, J. R. Larus, and D. A. Wood. Fine-grain access control for distributed shared memory. In *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VI)*, pages 297–306, Oct. 1994.

[12] D. A. Wood, S. Chandra, B. Falsafi, M. D. Hill, J. R. Larus, A. R. Lebeck, J. C. Lewis, S. S. Mukherjee, S. Palacharla, and S. K. Reinhardt. Mechanisms for cooperative shared memory. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 156–168, May 1993. Also appeared in CMG Transactions, Spring 1994.