Author Retrospective for
Anatomy of a Message In the Alewife Multiprocessor

John Kubiatowicz
Computer Science Division
University of California, Berkeley
kubitron@cs.berkeley.edu

ABSTRACT

The MIT Alewife project, launched in the Spring of 1988, comprised a dynamic group of researchers who designed and implemented the Alewife multiprocessor [1]. One of the most important and unexpected outcomes of this project was the message-passing interface described in "Anatomy of a Message in the Alewife Multiprocessor," selected for this Retrospective. This interface was essential to the Alewife OS and runtime systems, was important for I/O, and enabled a new and innovative cache-coherence protocol, called LimitLESS [3]. It also enabled direct comparisons between shared memory and message-passing [4].

Original paper: http://dx.doi.org/10.1145/165939.165970

Categories and Subject Descriptors

B.4 [Input/Output and Data Communications]: Interconnections (Subsystems)—*interfaces*; C.2 [Computer-Communication Networks]: Network Architecture and Design—*Packet-switching networks*

Keywords

Parallel computing, message-passing interfaces

1. INTRODUCTION

The MIT Alewife project was an amazing collaboration of students, research staff, and faculty that charted new ground, produced a wide variety of research results, and ultimately built real parallel processing hardware. It spanned the 10-year period from the Spring of 1988 to Spring of 1998—producing almost 30 papers in conferences and journals and 7–10 PhD theses (depending on how you count them). Lead by Anant Agarwal from MIT, the research environment was exciting, occasionally contentious, and always productive. As one of the principle architects of the Alewife hardware, I had the privilege to work directly with many outstanding researchers during the project. Together, we designed the Alewife machine from scratch with a new processor, new cache-coherence protocol, new compiler, novel synchronization support, and a custom operating system.

ICS 25th Anniversary Volume. 2014
ACM 978-1-4503-2840-1/14/06.
http://dx.doi.org/10.1145/2591635.2591650.

Much of the focus behind our papers was aimed at *practical* considerations that ultimately lead to the Alewife machine [1]. This "implementation realism" was one of our strengths as a research team. Whenever one of us would propose a new mechanism or protocol, the others would ask whether or not it was practical. During the course of the project, we constructed at least three different system-level simulators—including one called "the New World Order"—with the express purpose of understanding the complexity and performance of mechanisms under realistic loads.

One of the most important (and unexpected) outcomes of the early phase of the Alewife project was the message-passing interface. The final form of this interface was a collaboration between multiple members of the Alewife team. It rapidly evolved from a novelty to an essential mechanism.

2. THE INTERFACE IS BORN

When I first joined Alewife project in 1989, the team was investigating the design and implementation of limited directory cache-coherence protocols. Anant Agarwal "signed me on" to help design and implement the Alewife machine. At that time, there were some initial thoughts of what mechanisms should be in a good processor (ultimately leading to the April processor paper [2]) as well as a preference for packet-switched, low-dimensional networks over alternatives. The remainder of the details were in the very early stages and quite fungible. Since building a complete machine from scratch was an exciting prospect, I immediately began to flesh out the details.

The genesis for the Alewife message-passing interface was my desire to provide communication support for the operating system and I/O subsystem. In keeping with the fine-grained communication philosophy of Alewife, my initial proposal for the messaging interface included a low-overhead message description process, integrated direct memory access (DMA), and fast interrupt handling. At the time, Technology Square at MIT had a number of competing parallel computer projects, including the J-machine [8], Monsoon [9], and *T [7]. The J-machine, in particular, provided much inspiration to design a low-overhead message-passing interface that was compatible with our standard RISC pipeline[1]. I was quite familiar with the J-machine, since Bill Dally's research group was right next to ours.

As I recall, others in the group were extremely skeptical of my initial proposal. Many, including Daniel Nussbaum

[1]The J-machine included a specialized processor with hardware-support for method dispatch in object-oriented programming models such as Smalltalk.

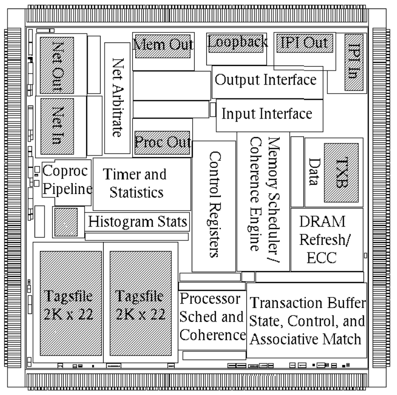

Figure 1: Floorplan for the Alewife CMMU (15mm $\times$ 15mm). Shaded regions are standard-cell memories. Remaining blocks are formed from sea-of-gates transistors.

Category	Gate Count	%	Mechanism			
			SM	MP	LT	FG
Processor Requests	11686	12	√		√	√
Full/Empty Decode	2157	2				√
Memory Machine	13351	13	√	√		
DRAM Control	8720	9	√	√		
Transaction Buffer	17399	17	√	√	√	
Tracking Vectors	2108	2	√		√	
Network Interface	11805	12	√	√		
Network Queueing	7363	7	√	√	√	√
CMMU Registers	9308	9	√	√		
Statistics	11958	12				
Miscellaneous	4627	5				

Figure 2: Functional block sizes (in gates) for the Alewife CMMU, as well as contributions to shared memory (SM), message passing (MP), latency tolerance (LT), and fine-grain synchronization (FG). Total chip resources: 100K gates and 120K bits of SRAM.

and David Chaiken, thought it was a distraction from our charter of making distributed shared-memory practical. I persisted, however, and had begun to think about the fact that "message-passing" (packet-switching) networks were going to be underneath any shared-memory implementation for Alewife. Soon, Anant Agarwal and David Chaiken realized that a low-overhead message-passing interface could be used to provide flexibility for the cache coherence protocol—allowing the common-case to be handled in hardware with special cases relegated to software. At that point, both the Alewife network interface and the LimitLESS cache coherence protocol [3] were born.

With the support of the Alewife team, I began designing the message passing mechanism in earnest—adapting it as necessary as others began applying it to their own interests. A flurry of activity followed. Specifically, David Chaiken began designing the LimitLESS coherence protocol; Ben-Hong Lim began designing synchronization mechanisms using messages; David Kranz and Dan Nussbaum began integrating messaging into the compiler and runtime system. Kirk Johnston, Ken Mackenzie, and Donald Yeung provided valuable insights at this time as well.

It is a testimony to our extensive simulation environment (complete with the ability to boot a binary version of our operating system) that we were able to quickly iterate on the design of the messaging interface.

3. A MATURE DESIGN

The "devil is in the details," as they stay. The "Anatomy of a Message" paper, included in this volume, is the result of an intense design process. It represents the (often colliding) combination of many design goals and implementation restrictions. I list a few of them here:

- The inclusion of the *Atomic Send* mechanism was necessary for reasonable integration with operating systems, since it allows the OS to interrupt user-level processes that are sending messages[2].
- The *High-Availability* interrupt mechanism was crucial to permit message-passing and shared memory to exist in the same machine.
- The *Locally-Coherent DMA* mechanism was a logical compromise between programmability and ease of implementation.
- Network overflow support (and corresponding queue topology) was essential, given that Alewife was implemented on top of a network with only a single virtual channel.

The strength of this paper arose, in my opinion, precisely because it was founded in real implementation concerns—without losing the simplicity of mechanism description. At the time that the paper was published, the Alewife communication and memory management unit (CMMU) was already well on its way toward implementation.

Figure 1 shows the final floorplan for the CMMU, illustrating where the message-passing mechanisms were placed. The queues and overall control interfaces (for the "Interprocessor Interrupt" or IPI) can be seen in the upper right corner of this chip. The upper left corner contains the network queuing and hardware interfaces to the network from the coherence protocol mechanisms.

This floorplan came directly from our LSI-Logic layout tool, and represents the final positioning of these components. Figure 2 shows that the message-passing mechanism comprises slightly more than 20% of the chip (it includes some of the CMMU registers), but that it impacts a number of important functions.

As a result of our extensive simulation and testing methodology (which included the ability to run a gate-level simulation of the CMMU in communication with a functional multiprocessor simulator), the first versions of the CMMU and Sparcle processor were substantially functional[3]. In fact, CMMU chips arrived in our lab on May 4th of 1994, and we had a 16-node Alewife machine running by June 17th.

[2]This is an important design flaw, in my opinion, of the J-Machine network interface.

[3]Many thanks to LSI Logic for their help in the design of the Sparcle processor and support with the layout and fabrication of the CMMU.

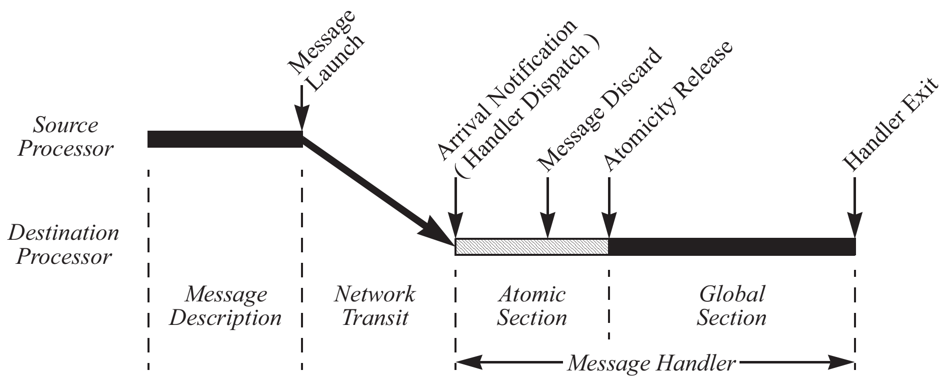

Figure 3: Injection of a User-Direct message and later extraction of this message with a user-level interrupt handler. A crucial aspect of this version of the message send was the *Atomic Section* which was entirely under control of the user-level task receiving messages.

4. EXTENSIONS TO THE SCHEME

The first version of the Alewife message-passing interface had some minor deficiencies with respect to user-level access to the network. Missing from that version was support for user-level control of message interrupts. As a result, Ken Mackenzie and I collaborated with Anant Agarwal and Frans Kaashoek to develop what we called "User-Direct Messaging" [6]. Support for this style of communication appeared in the second iteration of the CMMU.

User-Direct Messaging introduced a novel atomicity mechanism that allows user-level programs to disable message-arrival interrupts as long as they do not "abuse" the privilege; abuses include refusing to consume messages or consuming them too slowly. Figure 3 shows the complete life of a message as embodied in this new interface. Of particular importance to this figure is the *Atomic Section* that is entirely under control of the user.

5. 20 YEARS LATER

I like to think that everyone in the Alewife group eventually came to love the inclusion of a low-overhead message-passing interface into the Alewife universe. It altered our thinking about communication mechanisms significantly, changing the way we constructed essential software services [5] and spurring direct comparisons between message-passing and shared-memory communication styles [4]. The LimitLESS cache coherence protocol [3] immediately garnered attention from a wide audience and subsequently inspired a variety of follow-on work (both inside and outside of MIT) on the benefits of hardware invoked, software-supported cache-coherence protocols.

To this day, I believe that the Alewife message-passing interface is one of the most efficient and practical messaging interfaces ever designed. Its unique features are thanks to a dynamic group of researchers on the 6^{th} floor of MIT's technology square—a group of which I had the distinct privilege to be a member.

6. REFERENCES

[1] Anant Agarwal, Ricardo Bianchini, David Chaiken, Kirk Johnson, David Kranz, John Kubiatowicz, Beng-Hong Lim, Kenneth Mackenzie, and Donald Yeung. The MIT Alewife Machine: Architecture and Performance. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, June 1995.

[2] Anant Agarwal, Beng-Hong Lim, and David Kranz. April: A Processor Architecture for Multiprocessing. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, June 1990.

[3] David Chaiken, John Kubiatowicz, and Anant Agarwal. LimitLESS diectories: A scalable cache coherence scheme. In *Proceedings of the 22nd Annual Conference on Architectural Support for Programming Languages and Operating Systems*, April 1991.

[4] Fredric T. Chong, Rajeev Barua, Fredrik Dahlgren, John Kubiatowicz, and Anant Agarwal. The Sensitivity of Communicaton Mechanisms to Bandwidth and Latency. In *Proceedings of the Fourth Annual Symposium on High-Performance Computer Architecture*, February 1998.

[5] David Kranz, Kirk Johnson, Anant Agarwal, Beng-Hong Lim, and John Kubiatowicz. Integrating Message-Passing and Shared-Memory: Early Experience, May 1993.

[6] Kenneth Mackenzie, John Kubiatowicz, Matthew Frank, Walter Lee, Victor Lee, Anant Agarwal, and M. Frans Kaashoek. Exploiting Two-Case Delivery for Fast Protected Messaging. In *Proceedings of the Fourth Annual Symposium on High-Performance Computer Architecture*, February 1998.

[7] R. S. Nikhil, Greggory Papadopoulos, and David Culler. *T: A Multithreaded Massively Parallel Architecture. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, May 1992.

[8] Michael Noakes, Deborah Wallach, and William Dally. The J-Machine Multicomputer: An Architectural Evaluation. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, May 1993.

[9] Gregory Papadopoulos and David Culler. Monsoon: An Explicit Token-Store Archiecture. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, June 1990.