The Alewife CMMU: Addressing the Multiprocessor Communications Gap*

John Kubiatowicz, David Chaiken, and Anant Agarwal
with
Arthur Altman, Jonathan Babb, David Kranz, Beng-Hong Lim,
Ken Mackenzie, John Piscitello, and Donald Yeung

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

Company X has just expended 50 engineers over the last three years to produce their latest microprocessor. Now what? Well, popular wisdom suggests that they should connect a "bunch" of the micros together with some generic network to form a multiprocessor. This will yield "truly impressive performance" which the marketing department can quantify by multiplying the number of processors per box by the stellar MIPS-rating of the new microprocessor. Right?

Unlikely. Communication is fundamental to multiprocessing. In fact, the mechanisms for communication can make or break the performance of a multiprocessor. Consequently, "truly impressive performance" across a wide range of applications can be achieved only with careful design of the communication mechanisms.

The Alewife Solution. The Alewife Communications and Memory Management Unit (A-1000 CMMU) provides efficient, low-overhead communication mechanisms for general-purpose large-scale multiprocessing. It integrates both message passing *and* shared memory in a single unified framework. It can handle system configurations up to 512 processors, with each node in such a system consisting of an A-1000 CMMU, a Sparcle processor[1], an FPU, 64K of cache, 8Mb of DRAM, and an MRC network chip. The Sparcle processor is tightly coupled with the A-1000 CMMU[2]; however, this coupling is achieved through minor modifications to a standard SPARC implementation. Communications functionality of the A-1000 CMMU includes:

- Support for distributed, cache-coherent shared memory via the LimitLESS cache-coherence protocol[3]: the A-1000 supports up to five hardware pointers per memory line for normal data sharing and can invoke software interrupt handlers to employ additional pointers. Clean data can be fetched from a neighboring node in 30 cycles.
- Support for fast user-level messaging with integrated DMA[4]. A simple message, consisting of a header and one data word, can be launched in seven cycles.
- Several mechanisms for latency tolerance, including non-binding software prefetch and rapid context switching. A remote cache miss is signaled immediately to the Sparcle processor, which can switch to a new context in 14 cycles.

*To appear in HOTCHIPS '94.

In addition, the A-1000 CMMU includes a number of features for support of a complete multiprocessor system:

- A lockup-free cache controller with on-chip cache tags.
- A DRAM controller with refresh, ECC detection and correction, and ECC sweep.
- An interrupt controller for rapid shuffling of interrupt priorities.
- Asynchronous interface and queueing logic for the Caltech MRC network routers.
- An on-chip timer, cycle-counter, and full suite of statistics gathering facilities.
- Full support for fine-grained synchronization. Each 32-bit data word has an associated *Full/Empty* bit. A number of atomic actions can be performed on these bits, and they can be used to invoke traps on synchronization failures.

The A-1000 CMMU and companion Sparcle processor are part of a research effort to explore mechanisms for multiprocessing; they are *not* intended as commercial products. For expedience, most of the internal logic of the A-1000 CMMU was implemented with a high-level hardware synthesis language (called LES from LSI Logic). As a consequence, first silicon for the A-1000 CMMU is expected to run no faster than 33Mhz. Fabrication is currently in progress through LSI Logic.

Integrated Coherent Shared Memory and Message Passing. One of the salient aspects of the A-1000 CMMU is its smooth integration of shared memory and message passing in a single hardware framework. *Shared memory* refers to the presence of a global shared address space which all processors can access through load and store instructions. With shared memory, processors communicate by reading and writing to prearranged addresses in the global address space. *Message passing*, on the other hand, refers to the ability to send and receive explicit messages. It is important to note that the hardware unification of these two communication mechanisms is natural, since most modern implementations of shared memory make use of an underlying message passing network.

Shared memory provides a convenient abstraction for the expression of algorithms and as a target of compilation. Many algorithms are best expressed in a shared memory programming style but have irregular communication patterns which are not easily extracted by the compiler. Other algorithms share data at an extremely fine granularity, such that the software overhead of explicit message passing is prohibitive.

Consequently, some hardware support for shared memory is desirable. In the A-1000 CMMU, this includes hardware-managed caches for exploiting temporal and spatial locality within shared data references. Caches introduce the cache coherence problem, which is eliminated in the A-1000 CMMU via the LimitLESS cache coherence protocol (a unique combination of hardware and software mechanisms for cache coherence). Hardware support for shared memory in the A-1000 CMMU also includes rapid context switching for latency tolerance: whenever the CMMU initiates a long-latency operation, such as a remote cache fill, it signals the Sparcle processor through a synchronous trap; the processor can then switch rapidly to another context to begin performing useful work in another context.

While shared memory is advantageous in many situations, it is not always an appropriate communication paradigm. There are a number of situations in which direct, point-to-point communication through messages is desirable. These situations include operating systems functionality such as inter-processor interrupts, fast task dispatch, block I/O, and system-wide synchronization. Furthermore, when an application has sufficiently regular communication patterns, the compiler can extract and orchestrate communication through messages; this can be more efficient than relying on the communications heuristics employed by the cache coherence protocol.

The A-1000 CMMU provides a *user-level* message passing interface which is tuned for fast transmission and reception of both short messages (directly from registers) and long messages (including DMA). The efficiency with with short messages can be constructed is an important enabling factor in the LimitLESS cache coherence protocol. The two-phase *describe-and-commit* communication interface permits direct, user-level access to network hardware without impeding the use of this network by the operating system.

Hybrid architectures which provide both message passing and shared memory allow the best of both communications paradigms to be mixed and matched. The Alewife architecture, including the A-1000 CMMU, is one of the first to present such a unified hardware framework. Ongoing research is exploring the use of compiler technology to choose an optimal tradeoff between message passing and shared memory.

Physical Attributes. The A-1000 CMMU consumes approximately 90,000 gates of random logic and 100,000 bits of SRAM. It is implemented in LSI Logic's 300K hybrid gate-array process with 3 layers of metal; this technology provides standard cells for memory and a sea-of-gates for random logic. Built in scan provides visibility to most of the random logic during test; the remainder of the logic as well as the memories are tested through special interfaces. All of this is packaged in a 299-pin PGA.

At the time that this was written, fabrication of the A-1000 CMMU was in progress. Prototypes are expected in April 1994. Packaging for 16 and 128 node Alewife machines is ready.

The Alewife project is funded in part by DARPA contract # N00014-87-K-0825, by NSF grant # MIP-9012773, and by an IBM graduate fellowship. LSI Logic provided support for the fabrication of Sparcle, and partial support for the fabrication of the A-1000 CMMU.

References

[1] Anant Agarwal, Johnathan Babb, David Chaiken, Godfrey D'Souza, Kirk Johnson, David Kranz, John Kubiatowicz, Beng-Hong Lim, Gino Maa, Ken MacKenzie, Dan Nussbaum, Mike Parkin, and Donald Yeung. Sparcle: Today's Micro for Tomorrow's Multiprocessor. In *HOTCHIPS*, August 1992.

[2] Anant Agarwal, John Kubiatowicz, David Kranz, Beng-Hong Lim, Donald Yeung, Godfrey D'Souza, and Mike Parkin. Sparcle: An Evolutionary Processor Design for Multiprocessors. *IEEE Micro*, 13(3):48–61, June 1993.

[3] David Chaiken, John Kubiatowicz, and Anant Agarwal. LimitLESS Directories: A Scalable Cache Coherence Scheme. In *Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IV)*, pages 224–234. ACM, April 1991.

[4] John Kubiatowicz and Anant Agarwal. Anatomy of a Message in the Alewife Multiprocessor. In *Proceedings of the International Supercomputing Conference (ISC) 1993*, Tokyo, Japan, July 1993. IEEE.