StarT-Voyager*

Boon S. Ang and Derek Chiou
hahaha@lcs.mit.edu, derek@lcs.mit.edu
NE43-205, NE43-203

Abstract

This paper describes the StarT-Voyager Cluster Area Network (ClAN) Parallel System. ClAN is a new class of parallel systems designed to bridge the performance and functionality gap between traditional MPPs and NOWs (networks of workstations). By constructing a parallel machine with unmodified commercial SMPs as the building block, a ClAN system inherits most of the advantages of the NOW approach. In addition by chosing the coherent memory bus as the point of interface, a ClAN system achieves better message passing performance than NOWs, and also offers coherent distributed shared memory.

1 Introduction

StarT-Voyager is a prototype Cluster Area Network (ClAN) parallel System under development at MIT. We are proposing ClAN systems as a new class of machine to bridge the performance and functionality gap between MPPs / supercomputers, and networks of workstations (NOWs). The goal of StarT-Voyager is to turn a collection of commercial SMP's into a distributed parallel machine by replacing one of the processor cards in each SMP with a Network Endpoint Subsystem (NES) adaptor card. The SMP's are otherwise unmodified. StarT-Voyager will support coherent distributed shared memory and fast user-level message passing using a small but comprehensive set of mechanisms implemented by the NES hardware and system software.

2 Shared Memory Support

The shared memory support in StarT-Voyager builds on earlier works including the Stanford DASH[5, 6], Wisonsin Typhoon[8], MIT Alewife[1], Stanford FLASH[3] and earlier versions of StarT[7, 2]. As a prototype, StarT-Voyager is designed to efficiently support a wide range of experimentation, including studying a diverse set of coherent shared memory schemes. Central to this flexibility is the use of an embedded processor, which we call the sP (service processor), to control and coordinate coherence protocol actions. This is similar to the protocol processor in the MAGIC chip in FLASH. Although generally programmable, the sP should be viewed as running firmware code to avoid protection issues. In order for the sP to participate in shared memory accesses, the NES hardware provides it with two major sets of functionalities: (i) an interface between the sP and the aP-bus, and (ii) an interface between the sP and the network.

The NES interface to the aP-bus makes aP-bus transactions that are related to shared memory visible to the sP, and when desired, it can place the outcome of these transaction under the sP's control. The latter is required if the sP has to fetch data or carry out other coherence actions before allowing the transaction to complete. The interface also allows the sP to initiate transactions on the aP-bus, making it possible for the sP to initiate coherent actions to the local caches, and to transfer data between local DRAM and the NES.

Flexibility is designed into the NES aP-bus interface to accomodate two different ways of mapping shared memory addresses. One is the traditional shared memory machine approach, found on machines like DASH and Alewife, where the physical shared address space is meaningful across the entire machine and the NES is responsible for servicing accesses to addresses that are located on remote sites. In this capacity, the NES behaves as the slave for these addresses. A second view proposed in Typhoon and S-COMA[9] "demand maps" part of the shared memory space into local DRAM, essentially using it as a cache of shared memory, with page-level allocation, and cache-line granularity access control. The NES behaves as a snooper for these addresses. Special snooping hardware is provided in StarT-Voyager to speed up these snoop checks. This hardware facility is configurable, making it possible to experiment with fairly different protocols without sacrificing performance.

3 Message Passing Support

Both the sP and the aP are provided with a rich set of message passing functionalities. These functionalities include (i) DMA, (ii) medium sizes messages, (iii) short messages, and (iv) short messages with tag-on data. The DMA hardware performs transfer of data from DRAM at one site to DRAM at another site, and takes care of everything including reading data and packetizing it at the source end, writing data and keeping a count of the packets received at the receive end. For protection reasons, user-level aP DMA requests are directed to the sP which performs address translation and issues commands to the DMA hardware. Medium size messages can carry up to 88B of payload and allow message buffers, both for transmit and receive, to be cached. Short messages are optimized for very short messages containing 5B of payload. Finally short messages with tag-on data augments short-messages to allow a cache-line worth of the data in NES SRAM to be appended by the NES hardware to the message.

The NES hardware directly supports a small number of *resident* message queues (resident queues). These queues are buffered in NES SRAM and are managed by dedicated NES hardware, without sP intervention. User application code can directly access the resident message queues, just like in the CM-5[4]. Additional queues, which are buffered in DRAM and have exactly the same software interface as the resident queues, are managed by the sP. For these additional queues, the sP must explicitly move the composed messages from DRAM into a resident queue for transmitting and does similarly for received messages.

Most messages specify a logical destination which is translated by the NES hardware into a physical destination and a receive queue identifier. Only system code is allowed to

*This work is supervised by Prof. Arvind and is supported in part by the Advanced Research Projects Agency of the Department of Defense under the Office of Naval Research contract N00014-92-J-1310.

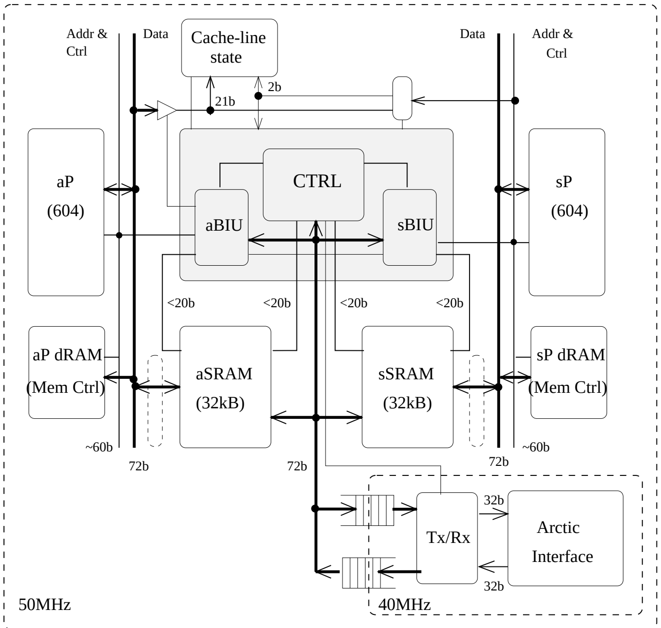

Figure 1: A StarT-Voyager site

modify the translation table. Therefore, system code specifies which destination nodes and which destination receive queues a particular process can send to, thus providing a level of protection.

4 Micro-Architecture

Figure 4 shows the micro-architecture of StarT-Voyager's NES. The NES board consists of (i) two banks of dual-ported SRAM (aSRAM and sSRAM), (ii) an ASIC (NESChip) which provides the interface to the 604's and controls the NES board, (iii) the sP sub-system which includes the sP (another 604), a memory controller and DRAM, (iv) an interface to the Arctic network (TxU/RxU) and (v) an SRAM to provide S-COMA cache-line permission bits.

The dual-ported SRAM's provide buffering for the message queues as well as space for the translation table, queues for communication between the aP and sP and buffer space for shared memory data. The NESChip has three basic components: the aP Bus Interface Unit (aBIU), the sP Bus Interface Unit (sBIU), and the Control Logic (CTRL.) The two BIU's provide the interface between their respective 604 address buses and CTRL.

The NES design tries to strike a balance between generality and performance. The inclusion of the sP allows for generality in the design through its programmability and its ability to control the NES hardware. To achieve the highest performance, dedicated hardware is provided for the most commonly executed NES functionality. In the cases where the sP is involved, the NES also provides a set of primitives that allow the sP to move blocks of data with a single command.

5 Implementation

We are currently in the process of implementing a 32-site StarT-Voyager. We plan to have a small prototype of 4-sites built by the end of this year. Except for the NESChip which will be implemented in a small laser-programmable ASIC, the NES board components will be entirely off-the-shelf. A team of graduate students and staff will be doing the implementation.

References

[1] A. Agarwal, R. Bianchini, D. Chaiken, K. L. Johnson, D. Kranz, J. Kubiatowicz, B.-H. Lim, K. Mackenzie, and D. Yeung. The MIT Alewife Machine: Architecture and Performance. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, 1995.

[2] D. Chiou, B. S. Ang, Arvind, M. J. Beckerle, A. Boughton, R. Greiner, J. E. Hicks, and J. C. Hoe. StarT-ng: Delivering Seamless Parallel Computing. In *Proceedings of EURO-PAR '95, Stockholm, Sweden*, pages 101 – 116, Aug. 1995.

[3] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chaplin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, and J. Hennessy. The Stanford FLASH Multiprocessor. In *Proceedings of the 21st Annual International Symposium on Computer Architecture, Chicago, Il*, Apr. 1994.

[4] C. E. Leiserson et al. The Network Architecture of the Connection Machine CM-5. In *Proceedings of the 1992 ACM Symposium on Parallel Algorithms and Architectures*, 1992.

[5] D. Lenoski, J. Laudon, K. Charachorloo, A. Gupta, and J. Hennessy. The Directory-Based Cache Coherence Protocol for the DASH Multiprocesor. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 148 – 159. IEEE, 1990.

[6] D. Lenoski, J. Laudon, T. Joe, D. Nakahira, L. Stevens, A. Gupta, and J. Hennessy. The DASH Prototype: Implementation and Performance. In *Proceedings of the 19th Annual International Symposium on Computer Architecture, Gold Coast, Australia*, pages 92 – 103, 1992.

[7] R. H. Nikhil, G. M. Papdopoulos, and Arvind. *T: A Multithreaded Massively Parallel Architecture. In *The 19th Annual International Symposium on Computer Architecture Conference Proceedings*, May 1992.

[8] S. K. Reinhardt, J. R. Larus, and D. A. Wood. Tempest and Typhoon: User-Level Shared Memory. In *Proceedings of the 21st Annual International Symposium on Computer Architecture, Chicago, Il*, pages 325 – 336, Apr. 1994.

[9] A. Saulsbury, T. Wilkinson, J. Carter, and A. Landin. An Argument for Simple COMA. In *Proceedings of the First IEEE Symposium on High-Performance Computer Architecture, Raleigh, North Carolina*, pages 276 – 285, Jan. 1995.