The Aries Decentralized Abstract Machine (ADAM)

Andrew Huang

The Problem: Large systems are challenging to manufacture and to operate with high availability. The problem extends from the manufacturing yield and the reliability of individual chips to the integration of cabinets and cables, and it must be addressed as an integral part of any massively parallel architecture. This problem is confounded by the fact that the average service life of a massively parallel machine is fairly long, and maintaining a base of exact replacement parts becomes difficult as manufacturing processes evolve. Ideally, one would replace worn-out parts with newer, faster parts without having to recompile or even restart running applications.

Motivation: The high cost of ownership and diminishing performance per dollar of full-custom, massively parallel computers has stymied research and development in this area; it seems that the classic big-iron machines has succumbed to the attack of the killer micros. However, here in the future of state-of-the-art pure-play silicon foundries, there is less of an economic tie between an architecture and a process technology. In addition, silicon is becoming cheap enough relative to the cost of packaging and testing that providing redundancy at the chip-level to increase yield and reliability is an attractive possibility.

Previous Work: There is an extensive body of previous work in the arena of high performance computing. The specific seminal works that this architecture stems from include decoupled access/execute machines [7], dataflow machines [3], processor-in-memory/chip multi-processor machines [5], named-state techniques [4], and architectures, languages and runtime environments that feature dynamic latency reduction and/or load balancing, such as CiLK [6], Active Threads [8], Threaded Abstract Machine [2], and COOL [1].

Approach: The Aries Decentralized Abstract Machine (ADAM) provides an abstract programming model (figure 1) that is independent of implementation details. It is in many ways similar to the java virtual machine concept, but it is tuned for high performance in a massively multithreaded environment with distributed shared memory. This layer of abstraction allows the underlying machine to be incomplete, imperfect, or inexact in many important ways.

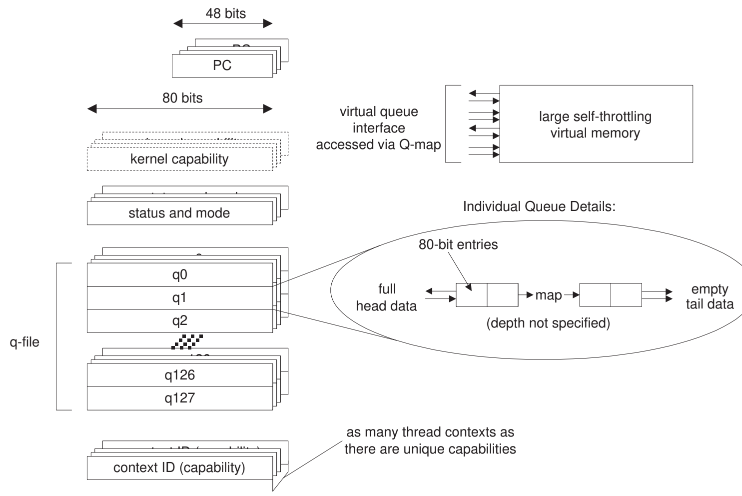

Figure 1: ADAM programming model

Like a JVM, an ADAM is started for each application that the user desires to run on the target machine. Thus, partitioning a job between several user jobs is a matter of distributing the available nodes between the ADAMs. Because the ADAM run-time model supports efficient dynamic data migration, load balancing between jobs can be done by forcing a job's objects to migrate out of some nodes and moving those nodes into other job's allocation pools. Please see the abstract entitled "Data and Process

Migration in the Aries Decentralized Abstract Machine (ADAM)" for more details.

Static failures due to manufacturing defects can be avoided by coloring the node allocation map. The basic structure of ADAM presumes a PIM-style chip-multiprocessor, so manufacturing yields can be improved because any die with at least one functioning processor could be considered salable die. Defects at the board and cabinet level can also be isolated at system integration time, allowing the machine to be come available before it is entirely assembled and debugged.

A machine that implements ADAM can be evolutionary. In its early stages, it may use emulation or emulation-assisted hardware. As the design is debugged and validated and as semiconductor processes improve, nodes are switched over to higher-performance dedicated hardware. This is done on the fly by forcing all data to migrate out of nodes to be serviced and bringing them off-line.

The ADAM also enables users to compensate for dynamic failures in a variety of ways. One can implement a voting scheme by running multiple identical ADAMs in parallel. Another approach utilizes early-warning hardware monitors to activate software that forces data migration out of a failing node. Also, ADAM's feature of isolating the entirety of a thread's state into a single capability simplifies the implementation of transactional rollback. The state of a thread before an operation can be copied to a backup location, which can be used as a rollback image in case the operation is aborted or failed.

Impact: Decoupling a programming environment from the details of the hardware implementation gives the system architect a powerful tool for dealing with cost of ownership issues and development issues. The incremental design methedology enabled by ADAM reduces time-to-solution and helps keeps development costs lower. Software teams developing compilers that target the abstract machine work concurrently with hardware teams that develop the custom silicon high-performance solution. Also, applications compiled for ADAM will run whether on a simulator, emulator, or final hardware.

Future Work: The system design concept enabled by ADAM is currently being validated by the concurrent Ansible hardware implementation effort and the Couatl compiler development effort. The ADAM specification is still evolving, as many significant issues such as the exception handling mechanism and architectural support for data migration are still under development.

Research Support: Support for this research was provided by the Air Force Research Laboratory, agreement number F30602-98-1-0172, "Active Database Technology".

References:

[1] Rohit Chandra, Anoop Gupta, and John L. Hennessy. Data locality and load balancing in cool. In *Proceedings of PPPP IV*, pages 249–259. ACM, 1993.

[2] David E. Culler, Anurag Sah, Klaus Erik Schauser, Thorsten von Eicken, and John Wawrzynek. Fine-grain parallelism with minimal hardware support: A compiler-controlled threaded abstract machine. In *Proceedings of ASPLOS IV*, pages 164–175. ACM, 1991.

[3] Ben Lee and A.R. Hurson. Dataflow architectures and multithreading. *IEEE Computer*, 1994.

[4] Peter R. Nuth and William J. Dally. The named-state register file: Implementation and performance. In *Proceeding of the First IEEE Symposium on High-Performance Computer Architecture*, pages 4–13, 1995.

[5] Kunle Olukotun, Basem A. Nayfeh, Lance Hammond, Ken Wilson, and Kunyung Chang. The case for a single-chip multiprocessor. In *Proceedings of ASPLOS-VII*, Cambridge, MA, 1996. ACM.

[6] Keith H. Randall. *Cilk: Efficient Multithreaded Computing*. PhD thesis, MIT, Department of Electrical Engineering and Computer Science, 1998.

[7] James E. Smith. Decoupled access/execute computer architectures. In *Proceedings of the 9th Annual International Symposium on Computer Architecture*, Austin, Texas, 1982.

[8] B. Weissman, B. Gomes, J.W. Quittek, and M. Holtkamp. Efficient fine-grain thead migration with active threads. *Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*, pages 410–414, 1998.