# Data and Process Migration in the Aries Decentralized Abstract Machine (ADAM)

Andrew Huang

**The Problem:** Access latency to memory is a performance bottleneck for contemporary computer systems. The partitioning of computers into separate memory and processor components has become a bottleneck simply due to the distance signals have to travel between these components. Figure 1 [1] illustrates the negative trend between on-chip wire scaling and the number of reachable bits within a fixed number of gate delays. Also, the complex memory hierarchy required to hide memory access latency hinders fast multiprocessor synchronization, limiting system scalability.

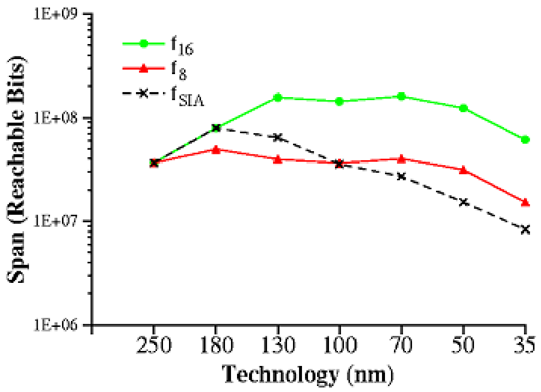


Figure 1: Span of reachable bits vs. wire geometry; a bit is 1 SRAM cell measured at 700 $\lambda^2$ (no SRAM support circuits considered). The span first increases for f16 (f16=16FO4) as the capacity of the chip increases. Later, as wire delays kick in, span decreases.

**Motivation:** The scientific and research communities have always demanded larger and faster machines. Many of the classic "big-iron" challenges have yet to be solved, and as machines become more powerful, computer modeling as a research tool has become more popular for almost every discipline. Some examples of problems that could benefit from bigger computers include computational chemistry, finite element modeling, large database systems, speech understanding and machine learning.

**Previous Work:** A large concern in parallel architectures that rely on caches for latency hiding are the false sharing, aliasing and capacity issues that lead to lower cache hit rates. The ADAM architecture distributes processors throughout memory at a fine granularity so there is less of a need for caches, and so the issues surrounding data placement and movement are different. The more relevant bodies of work are in the areas of process and thread migration. Active Threads [3] and Hsieh's thesis [2] on dynamic computation migration are examples of previous work that have addressed the issue of process and thread migration.

Please see also the abstract on the Aries Decentralized Abstract Machine (ADAM) for a broader overview of the ADAM architecture.

**Approach:** Seymour Cray once said "You can't fake what you don't have". If latency hiding is running out of steam, then latency *reduction* is the only option. In a classic centralized computer organization, overall latency is fundamentally limited by the amount of memory that can be packed within a certain clock-cycle radius of the processor. However, ADAM is a decentralized machine, *i.e.*, it is a sea of interconnect with small islands of processor and memory sprinkled throughout. In such a machine model, one reduces latency by migrating a process and its data closer together (figure 2).

Process and data migration are not new ideas; however, most efforts have been stymied by the overhead associated with migration. The limiting factors on migration include raw network performance, operating system overhead (updating process and routing tables, message passing overhead, etc.), and most importantly, pointer updates. Programs written in languages such as C or C++ are tough to break

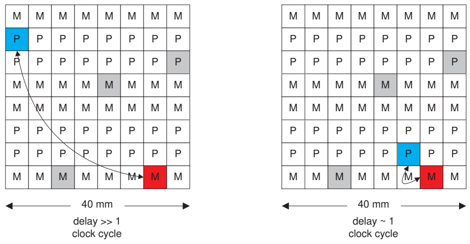


Figure 2: Illustration of process migration to reduce latency. Interconnect area omitted for clarity. Gray areas are defective nodes.

apart and migrate across a distributed machine due to the difficulty associated with tracking down the pointers to any given data structure. The stack-and-heap paradigm of a program runtime also complicates data relocation because of the complex pointer graphs within and between the stack and the heap.

The ADAM architecture provides an effective solution to this problem as a result of its capability-based addressing scheme, queue-based mechanism for passing values between threads, and global uniform address space for both memory and synchronization. Threads are light-weight under ADAM; they are tantamount to a method invocation in conventional systems. The entire state of a thread's computation–including the backing store for the processor state during a context switch–is encapsulated within a single capability that is also the thread's process ID. These IDs are unique throughout the machine. Architecturally visible queues that take the place of a register file in a conventional architecture enable communication between threads through queue remapping. By intimately integrating the inter-process communication and synchronization mechanism into the architecture, OS overheads and message passing overheads are kept to a minimum. Every queue mapping has the source and destination process IDs recorded with it, so that migrating a thread consists of simply copying the thread capability to a new location and updating the mappings, or leaving forwarding pointers where convenient. The decision to migrate a thread is done through an introspective mechanism implemented by a distributed management coprocessor.

**Impact:** The ability to efficiently and easily migrate objects throughout a machine creates a path for machine scalability in the latency-bound technology scenario of tomorrow. Most problems exhibit some locality, and object migration allows the run-time to exploit this locality without programmer knowledge of the exact size or physical layout of the target machine.

**Future Work:** A number of interesting and challenging issues have yet to be addressed in this nascent work. When and where to move objects is an important issue that has yet to be answered. The current idea is to provide an introspective distributed management coprocessor that runs in parallel with the ADAM execution cores; this coprocessor will take note of the amount and destination of network traffic, and make decisions to migrate processes based on these profiles. A first-pass at the migration algorithm is to use the "drift versus diffusion model" used to model minority carriers in semiconductor physics; the drift force would be proportional to the volume of communication per direction, and the diffusive force would be proportional to the congestion and load on individual nodes. This model was chosen because it demonstrates global self-organizing behavior through a set of local rules.

**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] William J. Dally (chair). The last classical computer isat study. Slides obtained electronically, 2001.

[2] W.C. Hsieh. Dynamic computation migration in distributed shared memory systems. Technical Report MIT/LCS/TR-665, Massachusetts Institute of Technology, September 1995.

[3] B. Weissman, B. Gomes, J.W. Quittek, and M. Holtkamp. Efficient fine-grain thead migration with active threads. *Proceedings of the First Merged International PPS and SPDP*, pages 410–414, 1998.