SPINE: Executing Application Code on I/O Processors

Marc E. Fiuczynski
Brian N. Bershad
Computer Science and Engineering
University of Washington
Seattle, WA

Richard P. Martin
David E. Culler
Electrical Engineering and Computer Science
University of California at Berkeley
Berkeley, CA

Input/Output intensive applications such as multimedia clients, file servers, and host based IP routers often move large amounts of data between devices, placing high I/O demands on both the host operating system and I/O subsystem. Although technology trends point to continued increases in link bandwidth, processor speed, and disk capacity, the lagging performance improvements and scalability of standard I/O interconnects (e.g. PCI) is increasingly becoming apparent for these applications.

Due to pressures for increased cache capacity and the standardization and longevity of I/O interconnects, we do not expect that I/O functionality will ever be integrated into general-purpose host CPUs. Therefore, we envision an architectural organization using I/O processors. This allows programmers to work around I/O limitations by trading cycles on the I/O processor for reduced interconnect crossings. The recent introduction of cheap, highly integrated I/O processors (e.g. StrongArm, MIPS, PowerPC) provide tantalizing building blocks for constructing systems to examine this tradeoff.

A key architectural concept that a system with I/O processors can exploit is the *inverted memory hierarchy*. Normally, access time increases down levels of the memory hierarchy. On an integrated I/O processor the normal memory hierarchy is inverted: access to the device media is cheap and fetching data from host memory and host CPU caches is expensive. Integration is important in realizing the inversion. Because the processor and I/O functions are on the same die, the I/O processor has excellent (even single cycle) access to the device media. However, the cost of integration is reduced speed, functionality, and cache capacity on the I/O processor. Yet our sample applications demonstrate that significant I/O performance improvements are obtainable due to integration and I/O specialization, even with slow I/O processors and limited capacity.

Using I/O processors is not a new idea; they have been around since the dawn of high performance computing. Both the CDC 6600 and IBM 360 architectures made extensive use of them. However, such I/O processors were primarily used to run operating system protocols (e.g. the file system and network stacks), limiting them to accelerating OS functions. Unfortunately, because OS protocols do not match application semantics well, high performance applications by-pass the OS as much as possible (e.g. raw-disk for databases, application-level framing for networks). Therefore, it was difficult for such applications to take advantage of I/O processors in the past.

We are investigating new approaches to programming I/O processors in the SPINE project[i]. We are working to move application-specific processing directly onto the network interface. Our approach is empirical: we take monolithic applications and migrate their I/O specific functionality into a number of *device extensions*. An extension is code that is logically part of the application, but runs directly in the network interface. The asymmetric nature of the SPINE model leads to an approach where the programmer looks for portions of the application where data movement into host memory is unnecessary or control transfers among devices are frequent. These pieces of code are good candidates to become extensions. Portions of the application that are compute intensive, or require complex interaction with the operating system should remain on the host system.

The SPINE software architecture offers the following three features that are key to the efficient implementation of I/O intensive applications:

- *Device-to-Device Transfers*: Avoiding unnecessary indirection through host memory reduces bandwidth requirements. Additionally, intelligent devices can avoid control transfers to the host CPU.
- *Host/Device Protocol Partitioning*: Servicing messages in the adapter reduces host overhead and latency, and the additional parallelism can increase message pipelining.
- *Device-level Memory Management*: Transferring data directly between buffers on the device and the application's virtual address space can improve system performance.

We use two technologies previously developed to realize SPINE: applications and extensions communicate via a message-passing model based on Active Messages, and, the extensions run in a safe execution environment that is derived from the SPIN extensible operating system.

Device extensions are expressed as a group of active message handlers, which are invoked when a message arrives for them. However, erroneous active message handlers should not stall forward progress of other messages or cause the system to crash. Consequently, it is necessary to guard against errant extension code running on the network adapter yet still maintain a low overhead execution environment. Our approach is a single address space design, where safety is achieved using a type-safe language, defensive interface design, and a combination of compile-time, run-time and download-time mechanisms. We limit the capabilities of extension handlers using language techniques, which enables SPINE to safely abort handlers that exceed their execution time allotment. As a direct result of SPINE's safety mechanisms, no context switches are necessary to run extension handlers, resulting in low overhead access to system services.

We demonstrate the validity of our approach with two sample applications: a video client and an Internet Protocol router. As a result of our system structure, image data is transferred only once over the I/O bus, directly from the network card into the frame buffer. The video client places no load on the host CPU, and is able to display at aggregate rates exceeding 100Mbps. The IP router can forward 11,000 packets per second on each network adapter using a 33 MHz "LANai" embedded processor. This rate is only 6% slower compared to a host-based implementation using a 200MHz Pentium Pro running FreeBSD. Like the video client, the SPINE IP router places no load on the host CPU.

Our two example applications indicate that high performance extensions are viable *even with an incredibly slow I/O processor*. A faster I/O processor, for example, would allow the use of a virtual machine interpreter (e.g., Java), enabling transparent execution of extensions regardless of the instruction set. Using a vector processor, as suggested by the IRAM project, as the I/O processor would enable data touching intensive applications, such as encryption, compression, video decoding, and data filtering, to be implemented on the network adapter.

---

[i] http://www.cs.washington.edu/research/spine