The Persistent Relevance of IPC Performance: New Techniques for Reducing the IPC Penalty

Wilson C. Hsieh, M. Frans Kaashoek, and William E. Weihl

{wchsieh, kaashoek, weihl}@lcs.mit.edu
MIT Laboratory for Computer Science

Although there have been substantial improvements in IPC performance (in particular, round-trip IPC time) in recent years, further reduction of the IPC penalty is still important for parallel applications and microkernel-based operating systems. We describe a number of new software and hardware techniques that we are investigating for reducing various components of the IPC penalty.

1. Introduction

Interprocess communication (IPC) is a valuable mechanism for structuring complex systems, as it allows systems to be decomposed into separate protection domains. In addition to being a good structuring mechanism, IPC is a fundamental primitive in distributed and parallel computing; the performance of distributed systems and parallel programs is often determined by the performance of IPC primitives.

Recent research has shown how to lower the cost of a round-trip IPC [Bershad et al. 1989, Druschel and Peterson 1992, Johnson and Zwaenepoel 1993, Karger 1989]. For example, a local RPC can be performed in 7.7 µs (254 cycles on a 33 MHz 486) [Yarvin et al. 1993], an RPC across an Ethernet can be performed in 340 µs (8800 cycles on a 25 MHz MIPS) [Thekkath and Levy 1993], and an RPC across an ATM link can be performed in 93 µs (2300 cycles on a 25 MHz MIPS) [Thekkath et al. 1993]. These numbers are remarkable compared to IPC performance in previous systems.

Although round-trip IPC time has improved dramatically, the IPC penalty is still significant, particularly in parallel applications and microkernel-based operating systems; we want to reduce the IPC penalty so that it is as close as possible to that of performing a procedure call. In Section 2 we discuss why IPC performance is important. In Section 3 we outline our ideas for reducing the IPC penalty, and in Section 4 we summarize our conclusions.

2. Why IPC performance is relevant

There are several reasons why IPC performance is important, and why further research is needed to improve it. Various trends are causing IPC to be used more frequently; in addition, research in other areas is reducing the overhead of other system bottlenecks. In this section we discuss these issues in detail.

Parallel computing requires high-performance IPC. First, workstation clusters are increasingly being used for parallel computing, since they are more cost-effective than custom multicomputers. Unfortunately, IPC performance in clusters is at least an order of magnitude worse than in custom multicomputers. To make distributed systems as attractive as parallel computers, a dramatic improvement in IPC performance is needed.

Second, parallel computers are being used more widely. Although inter-thread communication performance in custom parallel computers is very good, most multicomputers do not support multiprogramming well. As multicomputers become more widespread, users will demand that applications be protected from each other; this will lead to the addition of virtual memory (or at least virtual address spaces) to multicomputers. If it is not done carefully, IPC performance may be an order of magnitude more expensive than inter-thread communication.

IPC performance limits the decomposition of operating systems. Many production operating systems are structured around microkernels: e.g., Amoeba [Tanenbaum et al. 1991], Chorus [Rozier et al. 1988], Mach 3.0 [Accetta et al. 1986], Windows NT [Custer 1993], and QNX [Hildebrand 1992]. IPC performance is not yet good enough for these microkernels. For example, Mach 3.0 is often slower than Mach 2.5, which has a monolithic kernel [Anderson et al. 1991]; this performance difference is primarily due to the cost of more frequent IPC.

Several of the above operating systems avoid IPC because of its expense: Chorus allows services to be loaded into the kernel; Mach 3.0 has migrated some of the AFS cache manager back into the kernel [Nagle et al. 1993]; the Amoeba file server is usually configured to run in the kernel; Windows NT runs only non-critical services outside of the kernel. Others have argued that some server functions should be executed in client address spaces through the use of libraries [Bershad 1992], or that file systems should be in the kernel for performance reasons [Welch 1991]. We believe that IPC should not be avoided: if IPC performance were more efficient, we could afford to leave server functions in their own address spaces.

Device latency can be reduced or hidden. Some have argued that device latencies are a larger bottleneck than round-trip IPC time [Bershad 1992]. Current IPC mechanisms may be fast compared to devices, but the IPC penalty is still comparable to device access times; given the research trends in reducing or hiding the latencies of commonly accessed devices, this should remain true even as processors get faster. We will examine two examples, networks and disks.

Networks are getting faster; this is being driven by the convergence of parallel and distributed systems. In custom multiprocessors a great deal of money is invested in making networks fast [Agarwal et al. 1991, Leiserson et al. 1992]: network delays are on the order of a few tens of processor cycles. Mapping the hardware interface into registers or user space [Henry and Joerg 1992, Dally et al. 1991, Leiserson et al. 1992] can further lower the cost of sending messages. In distributed systems, network performance is approaching the performance of networks in custom multiprocessors. For example, the Fore ATM network interface (which can be mapped into user space) can transfer a 53-byte cell in 25 microseconds [Biagioni et al. 1993].

Although raw disk access times are not improving considerably, the latency of disk accesses can be hidden quite effectively. Recent advances in file system organization (e.g., log-structured file systems [Rosenblum and Ousterhout 1992]) allow data to be transferred to the file system at rates near the maximum bandwidth of the disk. In addition, the constantly decreasing price of RAM (current prices range from $35 to $65 per megabyte) will increase the size of file caches, which will help to hide disk latency. Finally, the increasing use of non-volatile RAM, which is also becoming steadily cheaper, will hide the latency of disks even more effectively [Baker et al. 1992].

IPC latency cannot always be hidden. Some might argue that IPC performance is unimportant because its cost can often be hidden by overlapping it with other computation. First, it is impossible to hide the latency of local IPC. Although it is possible to hide some of the latency of remote IPC, the primary cost of IPC is in software, which cannot be hidden. Second, none of the IPCs on the critical path of an application can be hidden: latency hiding may increase the throughput of a system, but it will not reduce the length of the critical path. Third, there may be insufficient parallelism available to hide IPC latency; this depends on the system and the applications. Finally, even if there were sufficient parallelism, a great deal of network bandwidth would be required to support it.

3. How to make IPC faster

In this section we describe our ideas for reducing the overhead involved with using IPC, which includes context switching, cache and TLB penalties, authorization and authentication, and control transfer. Some have argued that some of the above overheads, such as cache penalties, are more severe bottlenecks than simple round-trip IPC time [Bershad 1992]. We agree that cache penalties are certainly important bottlenecks, but we believe that all of the bottlenecks on the IPC path should be reduced together.

Manage the memory hierarchy in software. The memory hierarchy does not always handle memory-mapped devices efficiently. For example, network communication does not exhibit locality of reference: data in messages are often accessed only once. Such data is currently cached automatically, which causes cache lines to be wasted. It would be more efficient if there were mechanisms to avoid caching such references.

We are also exploring prefetching for reducing the cache and TLB penalties involved with IPC. There has been a great deal of research in cache prefetching for single applications (for example, [Mowry et al. 1992]), but we have not seen any work in prefetching across IPCs; similarly, there has been research in decreasing TLB penalties (for example, [Nagle et al. 1993]), but to the best of our knowledge no one has proposed prefetching for the TLB. Some architectures manage TLBs in software (e.g., MIPS [Kane and Heinrich 1992]); since the IPC path already goes through the kernel, prefetching TLB entries on such architectures should be straightforward. Since the IPC path is often predictable, it also should be straightforward to determine what cache and TLB entries to prefetch; we could use kernel-code synthesis [Pu et al. 1988] when the path is unpredictable.

Finally, the compiler can reduce context switching costs by managing register saves and restores across IPCs. Karger has proposed that a trusted linker be used to minimize saves and restores between domains that trust each other [Karger 1989]. We believe that more optimizations are possible for the compiler: for example, the compiler can hoist register saves and restores out of loops.

Generalize Active Messages. An Active Message [von Eicken et al. 1992] runs user code directly inside network interrupt handlers; it is a very efficient communication mechanism that eliminates the costs of thread creation, scheduling, and context switching. Active Messages are suitable for current multiprocessors, most of which do not support separate protection domains. We are extending the Active Messages techniques for IPC so that communication across protection domains is cheaper in parallel systems.

We are also extending Active Messages for use in multithreaded systems. First, we are studying how compilers can determine which procedures can safely execute as Active Messages; Active Messages that block on a lock or that execute for too long would cause serious problems. We are also investigating an approach we call "Optimistic Active Messages," in which most message handlers execute as Active Messages; if the handlers block or run for too long, which we expect to be uncommon, new threads are started for them. Using this approach, short non-blocking IPCs always execute as Active Messages, and blocking or long IPCs do not cause problems.

Support gang scheduling in distributed systems. Many parallel applications perform significantly better when components of the application run at the same time: processes that communicate frequently incur fewer context switches if they are gang scheduled. Such an approach involves changing network interfaces and operating systems in distributed systems, and is the approach taken in the CM-5 [Leiserson et al. 1992], which provides hardware support in the network for gang scheduling.

Adapt multithreading support for multiprogramming. The Alewife multiprocessor provides multiple register sets to enable fast switching of threads in a single address space [Agarwal et al. 1991]. Others have proposed using multiple register sets for fast context switching [Karger 1989], which would improve IPC performance; the SPARC Version 9 architecture allows register windows to be used for fast context switching [SPARC 1992]. We are examining how various ideas for improving the performance of multithreaded processors by reducing context switch costs and increasing utilization of register files can be adapted to improve IPC performance. Register relocation [Waldspurger and Weihl 1993] and the Named State Processor [Nuth and Dally 1991] are mechanisms for allowing threads to use a large register file flexibly; register dribbling [Soundararajan 1992] makes uses of excess bandwidth to memory to overlap computation and register saving. Extensions of these mechanisms for use across protection domains should be equally useful in multiprogrammed architectures.

Change network interfaces to support protection in hardware. The kernel currently authenticates and authorizes every IPC invocation. We propose having the sending network interface authenticate messages by tagging them with process identifiers that are set up at binding time, and having these identifiers authorized in the receiving network interface. Adding hardware in the form of special instructions may also be applicable to reducing the cost of local IPC [Karger 1989]; if possible, we would like to be able to eliminate the kernel from the IPC path.

4. Conclusions

Although round-trip IPC costs have been reduced dramatically in recent years, factors such as the increasing proliferation of microkernel-based operating systems still make IPC performance relevant. We have outlined several new software and hardware techniques for reducing the IPC penalty even further: manage the memory hierarchy in software, generalize Active Messages, support gang scheduling in distributed systems, adapt multiprocessor hardware ideas for reducing context switch costs, and support protection in network interfaces.

Acknowledgments

We would like to thank Henri Bal, Eric Brewer, Bob Gruber, Ulana Legedza, Brad Spiers, and Carl Waldspurger for their comments on various drafts of this paper.

References

Accetta, M.J., Baron, R.V., Bolosky, W., Golub, D.B., Rashid, R.F., Tevanian, Jr., A., and Young, M.W., "Mach: A New Kernel Foundation for UNIX Development," *Proc. Summer 1986 USENIX Conference*, pp. 93-113, Jul. 1986.

Agarwal, A., Chaiken, D., D'Souza, G., Johnson, K., Kranz, D., Kubiatowicz, J., Kurihara, K., Lim, B.H., Maa, G., Nussbaum, D., Parkin, D., and Yeung, D., "The MIT Alewife Machine: A Large-Scale Distributed-Memory Multiprocessor," *Proc. Workshop on Scalable Shared Memory Multiprocessors*, 1991. Extended version is MIT/LCS/TM-454.

Anderson, T.E., Levy, H.M., Bershad, B.N., and Lazowska, E.D., "The Interaction of Architecture and Operating System Design," *Proc. 5th Symposium on Architectural Support for Programming Languages and Operating Systems*, pp. 108-120, Santa Clara, CA, Oct. 1992.

Baker, M., Asami, S., Deprit, E., Ousterhout, J., Seltzer, M., "Non-Volatile Memory for Fast, Reliable File Systems," *Proc. 5th Symposium on Architectural Support for Programming Languages and Operating Systems*, pp. 10-22, Oct. 1992.

Bershad, B.N., "The Increasing Irrelevance of IPC Performance for Microkernel-Based Operating Systems," *Proc. USENIX Workshop on Microkernels and Other Kernel Architectures*, pp. 205-211, Seattle, WA, April 1992.

Bershad, B.N., Anderson, T.E., Lazowska, E.D., and Levy, H.M., "Lightweight Remote Procedure Call," *Proc. 12th Symposium on Operating Systems Principles*, pp. 102-113, Litchfield Park, AZ, Dec. 1989.

Biagioni, E., Cooper, E., and Sansom, R., “Designing a Practical ATM LAN,” *IEEE Network*, Vol. 7, No. 2, pp. 32-39, Mar. 1993.

Custer, H., “Inside Windows NT,” Microsoft Press, Redmond, WA, 1993.

Dally, W.J., Chao, L., Chien, A., Hassoun, S., Horwat, W., Kaplan, J., Song, P., Totty, B., and Wills, S., “Architecture of a Message-Driven Processor,” *Proc. 14th Annual International Symposium on Computer Architecture*, pp. 189-196, Pittsburgh, PA, Jun. 1987.

Druschel, P., and Peterson, L.L., “High-Performance Cross-Domain Data Transfer,” Dept. of Computer Science, Univ. of Arizona, TR 92-11, Mar. 1992.

Henry, D.S., and Joerg, C.F., “A Tightly-Coupled Processor-Network Interface,” *Proc. 5th Symposium on Architectural Support for Programming Languages and Operating Systems*, pp. 111-122, Boston, MA, Oct. 1992.

Hildebrand, D., “An Architectural Overview of QNX,” *Proc. USENIX Workshop on Microkernels and Other Kernel Architectures*, Seattle, WA, Apr. 1992.

Johnson, D.B., and Zwaenepoel, W., “The Peregrine High-Performance RPC System,” *Software Practice and Experience*, Vol. 23, No. 2, pp. 201-221, Feb. 1993.

Kane, G., and Heinrich, J., “MIPS RISC Architecture,” Prentice Hall, Englewood Cliffs, NJ, 1992.

Karger, P.A., “Using Registers to Optimize Cross-Domain Call Performance,” *Proc. 3rd Symposium on Architectural Support for Programming Languages and Operating Systems Principles*, pp. 194-204, Boston, MA, Apr. 1989.

Leiserson, C.E., Abuhamdeh, Z.S., Douglas, D.C., Feynman, C.R., Ganmukhi, M.N., Hill, J.V., Hillis, W.D., Kuszmaul, B.C., St. Pierre, M.A., Wells, D.S., Wong, M.C., Yang, S., and Zak, R., “The Network Architecture of the Connection Machine CM-5,” November 1992. An early version appeared in *Proc. 1992 Symposium on Parallel Algorithms and Architectures*.

Mowry, T.C., Lam, M.S., and Gupta, A., “Design and Evaluation of a Compiler Algorithm for Prefetching,” *Proc. 5th Symposium on Architectural Support for Programming Languages and Operating Systems*, pp. 62-73, Boston, MA, Oct. 1992.

Nagle, D., Uhlig, R., Stanley, T., Sechrest, S., Mudge, T., and Brown, R.,”Design Tradeoffs for Software-Managed TLBs,” *Proc. 20th Annual International Symposium on Computer Architecture*, pp. 27-38, San Diego, CA, May 1993.

Nuth, P., and Dally, W., “A Mechanism for Efficient Context Switching,” *Proc. International Conference on Computer Design*, pp. 301-304, Cambridge, MA, Oct. 1991.

Pu, C., Massalin, H., and Ioannidis, J., “The Synthesis Kernel,” *Computing Systems*, Vol. 1, No. 1, pp. 11-32, Winter 1988.

Rosenblum, M., and Ousterhout, J.K., “The Design and Implementation of a Log-structured File System,” *ACM Transactions on Computer Systems*, Vol. 10, No. 1, pp. 26-52, Feb. 1992.

Rozier, M., Abrossimov, V., Armand, F., Boule, I., Gien, M., Guillemont, M., Herrmann, F., Kaiser, C., Langlois, S., Leonard, P., and Neuhauser, W., “Chorus Distributed Operating System,” *Computing Systems*, Vol. 1, No. 4, pp. 305-370, Fall 1988.

Soundararajan, Vijayaraghavan, “Dribble-Back Registers: A Technique for Latency Tolerance in Multiprocessors,” MIT/LCS/TM-474, Oct. 1992.

SPARC International, “SPARC-V9 Architecture Specification”, Release R1.2, Nov. 2, 1992.

Tanenbaum, A.S., van Renesse, R., van Staveren, H., Sharp, G., Mullender, S.J., Jansen, A., van Rossum, G., “Experiences with the Amoeba Distributed Operating System,” *Communications of the ACM*, Vol. 12, No. 33, pp. 46-63, Dec. 1990.

Thekkath, C.A., and Levy, H.M., “Limits to Low-Latency Communication on High-Speed Networks,” *ACM Transactions on Computer Systems*, Vol. 11, No. 2, pp. 179-203, May 1993.

Thekkath, C.A, Levy, H.M., and Lazowska, E.D., “Efficient Support for Multicomputing on ATM Networks,” Dept. of Computer Science and Engineering, University of Washington, Technical Report 93-04-03, Apr. 1993.

von Eicken, T., Culler, D.E., Goldstein, S.C., and Schauser, K.E., “Active Messages: a Mechanism for Integrated Communication and Computation,” *Proc. 19th International Symposium on Computer Architecture*, pp. 256-266, Gold Coast, Australia, May 1992.

Waldspurger, W., and Weihl, W.E., “Register Relocation: Flexible Contexts for Multithreading,” *Proc. 20th Annual International Symposium on Computer Architecture*, pp. 120-130, San Diego, CA, May 1993.

Welch, B., “The File System Belongs in the Kernel,” *Proc. 2nd USENIX Mach Symposium*, pp. 233-250, Nov. 1991.

Yarvin, C., Bukowski, R., and Anderson, T., “Anonymous RPC: Low-Latency Protection in a 64-Bit Address Space,” *Proc. Summer 1993 USENIX Conference*, pp. 175-186, Cincinnati, OH, Jun. 1993.