Communication and I/O Libraries

Communication and I/O Libraries

Chair: S. Lennart Johnsson
Scribe: Patrick Worley

TR-02-91

February 1991

Parallel Computing Research Group

Center for Research in Computing Technology
Harvard University
Cambridge, Massachusetts

Air Force Office of Scientific Research grant: 44-752-9000-2.
National Science Foundation grant: 44-752-9705-2.

Workshop summary from the *DARPA Workshop on Scalable Libraries*, Oakridge National Laboratories, September 1990.

Communication and I/O Libraries

Chair: Lennart Johnsson

Scribe: Patrick Worley

October 16, 1990

Introduction

Standardized communication and I/O libraries are necessary tools in developing and maintaining scalable libraries. Scalable libraries must perform correctly and efficiently for a large range of problem sizes and for a diverse set of computer architectures. High performance architectures of the future will be characterized by a large number of processors and a physically distributed memory. High performance on computers with large numbers of processors requires efficient communication and I/O algorithms, which are necessarily architecture dependent. Communication libraries serve to shelter both users and developers of numerical libraries from data motion through the communication system without sacrificing performance. Libraries also enhance programming efficiency by reuse of tested and debugged code.

One of the most important purposes of a library is to enable programs to be written that are both portable and efficient across a range of architectures. But the number of processors and the nature of the interconnection network affect the choice of data structure that application programs use, and the data structures determine both the syntax and semantics of communication functions that deal with the data. Therefore, any standardized communication library must support a rich assortment of data structures and must represent a consistent design to which it is simple to add new data structures and communication functions.

Most programming languages are designed for a shared, global address space. It is generally agreed that such a treatment of memory is preferred whenever performance is not an issue. But, in distributed-memory machines, data allocation and data movement between processors are critical factors in determining performance. Some programming languages offer higher-level data manipulation functions that can be implemented efficiently even when the data structures are distributed across processors. Examples of such operations are reduction, copy, and reshaping. Inclusion of such functions in a communication library provides portability even between shared- and distributed-memory computers, and provides a simple programming environment that still supports efficient implementations for many applications.

In order to produce and maintain a communication library with the attributes discussed so far, it is important that a layered approach be used. The top layer should be independent of the computer architecture, but the lower level(s) may have a dependence upon the architecture and upon the programming model. Experience with current multiprocessors indicates that a relatively small set of low-level primitives may suffice. Higher-level communication functions can be created through composition of low-level functions.

I/O and secondary storage systems for multiprocessors are currently undergoing very rapid development, and little experience with such systems is available. While it is clear that I/O libraries are crucial to the efficient use of large multiprocessor systems, the specification of such libraries are premature at this time.

Specification of a Communication Library.

The purpose of a communication library is to allow application codes to be portable without sacrificing performance. The functions included in the library form an extension to the programming language being used to express the computations, and must be capable of dealing with the data structures provided by the language. The definition of a communication function is often partially determined by the structure of the data on which it is operating, especially when the data object is distributed across the multiprocessor. Some of the characteristics that affect the specification of a communication function are

- the data structures (array, linked list, tree, etc.) supported by the function. Note that the global and local data structures may have different properties.

- how the data structure is partitioned across the memory heirarchy. For example, if an instance of a data structure has several elements per processor or memory module, then common allocation schems are cyclic/wrap/scatter, consecutive/block, and random.

This characteristic is also often refered to as data layout, or data geometry.

- subselection/context (as in gather/scatter).
- reshape/equivalence. Data structures often need to be dynamic, but even when a static data structure is sufficient, programming convenience and memory requirements may require that a given data structure be viewed in several different ways. Reshaping serves this purpose. Communication functions must use this information for a correct result.
- the size of the data structures relative to the machine size.

The functions that need to be included in a communication library initially are a function of the needs of the important application codes. Some of the functions that frequently occur in application codes include

- remote assignment and references
- broadcast,
- global exchange,
- transposition,
- bit-reversal,
- vector-reversal,
- arbitrary permutations,
- random permutations,
- reduction with various operators (add, and, or, ...),
- scatter,
- shift,

- convolution,
- butterfly network emulation, and
- PM2I network emulation (cyclic reduction, divide-and-conquer, ...),

all of which must be qualified by each of the five characteristics described previously. The functionality of the lowest level of primitives will also depend upon the capabilities of the underlying hardware and system software. For example, the influence of the native mode of the communication system, such as packet or circuit switched, pipelined or not, blocking scheme, operating system overhead, etc., are likely to be important at the lower levels of the library. The implementations of the higher-level functions may also need to be machine specific when efficiency is required.

Missing Science

Communication libraries can be, and have been, designed and implemented for distributed memory architectures. There is no missing hardware technology preventing the design or implementation of such libraries. However, the following unresolved issues make the design of communication libraries difficult.

The lack of standards in any form has hindered codes that use libraries from being portable. Existing communication libraries are often specific to particular brands of distributed memory architectures, and portability of libraries across machines is often limited.

Deficiencies in existing compiler technologies increase the number of functions required in a communications library. This leads to the risk of a "combinatorial explosion" in the size of communication libraries that are designed to assure high system utilization for the data motion patterns used in application programs.

Limited understanding of efficient data motion in arbitrary networks may force a redesign of communication libraries should networks other than meshes, Boolean cubes, and butterfly networks be used in the future. The search for "universal" networks with good properties under a variety of conditions is still an active area of research.

The lack of language standards makes any design much more complicated. While there is little hope for a resolution of this issue in the near future, a decision on a reasonable subset of languages to support must be made at the outset of any design effort.

Studies are needed of the communication needs of existing large-scale applications and algorithms. If the design is done right, adding new functions to the library will be straightforward, but prioritizing the design of the functions to be included initially will help promote the utility of the library and encourage vendors to adopt it.

Finally, experience with I/O systems and display storage for large multiprocessor systems is sorely lacking. Until more is known on what is possible and how these systems will and should be used, design of the I/O component of any communication library will necessarily be significantly delayed.

Research Issues

Application codes designed for scalable high-performance computers must be investigated to determine what high-level functions are most important to implement initially, and what data structures, selection mechanisms, etc. must be supported. The library designers must worry about how to layer the primitives to build a library with a consistent design that is still easily extensible. Algorithm research is necessary to indicate how to implement most efficiently both the low-level and high-level functions. Research is also needed into how best to encapsulate machine parameters for the high-level routines, that is, how to guarantee efficient implementations while incurring as little machine dependence as possible. Research is required into the incorporation of instrumentation and debugging hooks, and into what level of error checking should be supported. Once a generally accepted form of the library is available, research can begin on what hardware and operating system support is most suitable of the agreed-upon functions.

Expectations

At the end of two years, we would like to see the specification of a nontrivial set of procedural communication primitives that are suitable for FORTRAN90 and programs in some equivalent C dialect, such as C++ or C*. Moreover, it is important that a prioritized list of high-level communication functions be compiled, with the design and implementation of the most important ones already begun.

As part of the research efforts during the first two years, case studies of existing application codes should be used to identify a significant number of high-level functions that are to be included in the library, and a consistent user interface should be designed. Moreover, efficient implementations of all primitives should be established, as well as for many of the important high-level routines. Research should identify how to layer the library to permit extensibility, and initial attempts to provide instrumentation and debugging hooks should be made.

At the end of five years, we hope that a vendor-independent standard communication library will have been established by consensus within the user community, with vendor support for the standard being vigorously lobbied for. By this time, research should have indicated what sort of hardware and/or operating system support is most appropriate for the library, and what tradeoffs must be taken into account. The library should continue to grow throughout these next three years as more high-level routines are included to take into account the results of further case studies, as well as feedback from the user community. New functions should also have been included to support communication functions across networks of heterogeneous machines. We also expect functions supporting I/O to have been identified as more experience with such peripherals becomes available.

Conclusions

A communication library is a necessary component of any scalable library. A standard set of interfaces is critical to enable high performance computing technology to evolve independent of the evolution in applications. A good design and efficient implementation of communication libraries is necessary for high performance computing technology to become accessible to scientists and engineers requiring large scale computation. The specification, design, and implementation of standard communication libraries is urgently needed, and should commence immediately.