Foreword

Software libraries are important tools in the use of computers. Libraries for scientific and engineering applications embody expert knowledge of data structures, algorithms, operating systems, compilers, computer architecture, applied mathematics, and numerical analysis. Libraries enhance productivity not only by providing preprogrammed functions, but, what is more important, by providing functions that have well-understood and documented storage requirements, execution time, and numerical behavior. Libraries make the architecture of computers more transparent to a user than programming languages by defining functions at a sufficiently high level for optimization beyond the capabilities of compilers. The portability of user programs is enhanced with respect to both performance and numerical behavior. Libraries substantially lower the cost of computation, improve productivity, and enhance the quality of the end result.

Developments in computer architecture represent particularly strong forces behind the evolution of libraries for high-performance computers. The new generation of high-performance architectures, scalable to several trillion operations per second, consists of thousands of processing units with local memories, and a network interconnecting the processor and memory units. The performance optimization of functions to be executed on such architectures requires careful attention to data allocation, data motion in distributed data structures, memory hierarchies, load balancing, and scheduling of pipelines. Fundamental changes of classical algorithms may be required. The efficient use of such scalable computer architectures is beyond the capability of state-of-the-art compiler technology.

Libraries provide significantly more powerful constructs than those available in most programming languages. The invocation of a library routine implies that a function be applied to the objects defined in a programming language, and that information about the objects be extracted, transformed, or used to generate new objects. The array syntax of recently introduced programming languages has a profound impact on the interface to a library and on its functionality and design. The array syntax of

Copyright © 1994 Thinking Machines Corporation

programming languages is in part motivated by the emergence of parallel computer architectures, particularly data parallel architectures. Concurrency occurs both in applying high-level functions to disjoint data sets, sometimes defined through a recursive procedure, and in each application of the function.

The Connection Machine Scientific Software Library (CMSSL) is created for languages with an array syntax and for data parallel architectures. The CMSSL is designed to handle concurrent application of a function to disjoint segments of arrays, and concurrent execution of each application. Concurrent application of the same function to segments of arrays implies computation on multiple instances, a very important feature for library routines on scalable architectures. The multiple-instance feature provides concurrency control for the library independent of the control structures in the language to which it is interfaced. The multiple-instance paradigm enhances portability and is a new feature for scalable architectures.

In the CMSSL, efficient use of the Connection Machine system architecture is accomplished through a careful choice of data layout, efficient implementation of interprocessor data motion, and careful management of the local memory hierarchy and data paths in each processor. The library accepts any data layout that can be specified for any machine configuration. Internally, library functions may reallocate arrays for optimum performance, or to establish a common processor configuration for all operands in a function evaluation. Performance tuning through control of the data allocation is largely new to data parallel architectures, though some of the issues are analogous to those occurring in banked memory systems and systems with a cache.

The CMSSL achieves architectural independence with respect to data motion through a set of communication functions providing a shared memory view of the global address space. Efficient management of the resources for each processor is achieved through level 2 and level 3 Basic Linear Algebra Subroutines (BLAS). Blocking schemes are used for some BLAS functions, and for functions such as the Fast Fourier Transform, for which high-radix algorithms are advantageous with respect to performance.

Version 3.2, April 1994
Copyright © 1994 Thinking Machines Corporation

In summary, the CMSSL is a library for languages with an array syntax and addresses many new issues related to concurrency control, data allocation and data motion in distributed data structures, language independence, and scalability. It is our hope that the CMSSL will serve the users of distributed-memory architectures well, and that it will evolve to include a broad set of basic functions frequently used in scientific and engineering applications, as well as higher-level functions for ordinary and partial differential equations, optimization, and signal processing.

S. Lennart Johnsson

Director of Computational Sciences, Thinking Machines Corporation
Gordon MacKay Professor of the Practice of Computer Science, Harvard University
Cambridge, Massachusetts
December 1992

Copyright © 1994 Thinking Machines Corporation