# The Connection Machine Systems CM-5

The Harvard community has made this article openly available. **Please share** how this access benefits you. Your story matters

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Citable link</td>
<td><a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:24829614">http://nrs.harvard.edu/urn-3:HUL.InstRepos:24829614</a></td>
</tr>
<tr>
<td>Terms of Use</td>
<td>This article was downloaded from Harvard University’s DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at <a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA">http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA</a></td>
</tr>
</tbody>
</table>
The Connection Machine Systems CM-5

S. Lennart Johnsson

TR-09-93
April 1993

Parallel Computing Research Group
Center for Research in Computing Technology
Harvard University
Cambridge, Massachusetts

The Connection Machine Systems CM-5

S. Lennart Johnsson
Thinking Machines Corp.
and
Harvard University
Cambridge, MA

Abstract

The Connection Machine system CM-5 is a parallel computing system scalable to T\text{flop} performance, hundreds of G\text{bytes} of primary storage, T\text{bytes} of secondary storage and G\text{bytes}/s of I/O bandwidth. The system has been designed to be scalable over a range of up to three orders of magnitude. We will discuss the design goals, innovative software and hardware features of the CM-5 system, and some experience with the system.

1 Design goals

The Connection Machine system CM-5, introduced in 1991, was designed to advance state-of-the-art in massively parallel processing by providing a productive programming environment, flexible configuration of processing and storage capacity and I/O bandwidth, easy integration into networks, convenient multiuser access, and high reliability and availability. Except for scalability, these features are taken for granted in most computer systems.

Scalability has both software and hardware implications. Scalability from a software point of view implies that a program shall be able to execute without change over a wide range of system sizes and yield correct results within acceptable tolerances. The efficiency for problem sizes scaling with machine size, shall be independent of the machine size, given a scalable hardware architecture. Achieving this goal has a significant impact on most software systems, in particular run-time systems and libraries.

Hardware scalability requires not only that sufficient processing and storage capacity can be added, but also that the I/O capacity and the data motion capacity can be scaled as well. The latter aspect has a fundamental impact on the communication system.

An effective multiuser environment requires that both time-sharing and partitioning of nodes into independent sets is supported. Partitioning allows jobs to run on an appropriate set of nodes, while time-sharing allows for increased resource utilization. Time-sharing a distributed memory system represents a unique challenge in that part of the user state may include messages in route between source and destination. Any multiuser system must also guarantee security of interference between jobs.

High availability is critical in a large-scale system. It is not sufficient to design a highly reliable system. Fast detection and isolation of faults is necessary as well as the ability to operate a system with some faulty components. High availability also requires that any component failure affects a very limited part of the system and that the rest of the system can remain in service while faulty parts are being replaced.

2 Software systems

2.1 Operating system

The operating system for the CM-5, CMost is a modified version of UNIX. The modifications serve to enhance the management of the CM-5 resources, such as internode communication and I/O, while standard UNIX features allows the CM-5 to be easily integrated into a heterogeneous, networked environment, provide time-sharing and system administration support, and protection between different user spaces. The enhancements provide: high speed access to a file system striped across a parallel disc system, checkpointing, fast internode communication, support for both data parallel and node level programming models, central administration and resource management.

CMost allows the system to be space-shared by configuring the nodes into independent partitions, each under the control of a partition manager that runs a full version of CMost. I/O control processors may also serve as partition managers. Time-shared access to partitions are facilitated through standard UNIX features. Users can use any partition through remote login, or through the batch processing system NQS (Network Queueing System). The CM-5 administration is centralized at a system console with commands modeled on SunOS 4.1.

All I/O operations are modeled as standard UNIX reads and writes to files, regardless of the device. CMost extends standard UNIX to parallel reads and writes, and to the access of very large files. CMost supports many different file system types, including standard UNIX file systems and NFS. Data is stored in canonical (serial UNIX) order. For parallel I/O, data is reordered between the canonical order and the desired parallel order. Thus, data can be written by a program running on a given partition, then read back with the programming running on a different size partition. Data can also be exchanged through sockets between a CM-5 process and a process running on a processor connected through external networks.

2.2 Programming environment

The CM-5 supports a Motif-based graphical programming environment, Prism. Prism supports program development, debugging, execution and performance tuning. It supports dbx-like debugging, graphical display of performance data and visualization of program variables.
2.3 Programming models

The CM-5 supports two programming models: data parallel and node level. The data parallel model is supported through Connection Machine Fortran, CMF; (to become High Performance Fortran, HPF) and C*. The node level programming model is supported through CMF and a message passing library CMMD. Node level programming can also be carried out in C and, but compiler support for these languages is currently not at the level of CMF and C*.

The data parallel model has a global address space. The supporting run-time system performs data allocation and reallocation as required by the user program. All communication required by references to remote data is handled by the run-time system transparently to the user code. No explicit data motion commands are required. Programs written in HPF (when supported) should run unchanged not only on other parallel computers, but also on any serial computer.

The effective use of memory hierarchies is critical for high performance. Compilers and run-time systems on conventional architectures today often do not achieve a good exploitation of locality of reference with carefully crafted codes. But, in distributed memory systems the software technology is much less mature. CMF, like HPF, provides the programmer with data management directives, which can enhance locality of reference when the programmer has knowledge of the data reference pattern.

In the absence of directives, the Connection Machine Run-Time System, CMRTS, manages data layout using two simple rules: 1) distribute the data as evenly as possible over all nodes (for sufficiently large data sets), 2) minimize the surface to volume ratio for any array. The first rule is based on the assumption that engaging as large a number of nodes in the computation as possible reduces the computation time. This assumption is appropriate for the CM-5.

The second assumption is sensible for computations such as explicit finite difference codes, convolution codes, and other codes for which the predominant data references are uniform with respect to the different axes.

In the case of computations involving multiple arrays of different shapes, such as matrix-vector multiplication, matrix-matrix multiplication, and the solution of systems of equations with one or several right hand sides, it is critical for performance that a suitable data allocation be chosen for the different operands, and that the required data motion be made as efficient as possible. The Connection Machine Scientific Software Library, CMSSL, not only chooses an optimal configuration of the collection of nodes with respect to execution time, but also manages the data motion efficiently through a set of optimized communication routines. These include all-to-all broadcast, polyshift, gather and scatter, axes exchange (matrix transposition), and a special convolution compiler. In addition, CMSSL routines also exploit the local memory hierarchy efficiently through two levels of two- or three-dimensional tilings of the index space local to a node. These techniques have resulted in peak efficiencies of up to 99.95% for the local BLAS. For global operations, peak efficiencies in the range 50 – 70% are achieved.

The message passing library CMMD supports both blocking and non-blocking passive messages and active messages, as developed at UC Berkeley. The CMMD library also supports some global operations, such as broadcast from host, reduction to host, reduction to all nodes, parallel prefix, segmented parallel prefix, and gather and scatter operations.

In the visualization area AVS is supported.

3 Hardware systems

The main innovative hardware features of the CM-5 are in the network and I/O subsystems. The CM-5 has three networks: the Data Network, the Control Network, and the Diagnostics Network. The CM-5 nodes currently are of four types: processing nodes, disc nodes, tape nodes, and I/O nodes.

The Data Network is a fat-tree network with a perfect scalability beyond the first two levels. Beyond those two levels, the bisection width grows in proportion to the number of nodes in the network. For the first two levels there is a factor of two reduction in the bisection width per node for each level. The network latency depends upon the size of the system, and is in the range 1 − 5 μsec. Randomized routing is used for load balance in the network.

The Control Network provides a means for very fast processor synchronization. It also supports parallel prefix operations, broadcasting and reduction.

The Diagnostics Network can probe and control the system. All CM-5 components are designed to be testable when in place in the system. Nearly all data paths are protected by parity or full CRC. All DRAM is protected by ECC for single bit error correction and double bit detection. Transfers through the Data and Control Networks are checked for each link. Failed components can be logically and electrically isolated from the rest of the system under the control of the Diagnostics Network.

The processing node architecture consists of a Sparc processor, four vector units, and a network interface. Each vector unit has its own memory bank accessed through a single 8 byte wide data path. Vector units operate at 16 MHz, while the Sparc operates at 32 MHz. Each memory bank contains 8 Mbytes of memory using 4 Mbit DRAM. The system is designed to be upgradable to 64 Mbit DRAM.

The vector units run a standard vector instruction set.

The disc nodes support a Scalable Disc Array, SDA. Each disc node offers 9.2 Gbytes of storage available at a rate of 12 Mbytes/s. The SDA is based on 3.5" drives. The CM-5 system can be configured with an arbitrary number of disc nodes. A transfer rate of about 1.3 Mbytes/s per Gbyte of disc storage is maintained.

I/O nodes can be ganged together for HIPPI channels, or used directly for network connections of speeds of up to 20 Mbytes/s.

4 Experience

The CM-5 system has been available for close to two years. As a validation of the scalability goal we conclude with performance data for the Linpack benchmark using CMSSL, and for some finite element computations on unstructured grids. The performance data below applies to 64-bit IEEE precision.

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>Linpack peak</th>
<th>FEM unstruct</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>0.98</td>
<td>0.79</td>
</tr>
<tr>
<td>32</td>
<td>1.94</td>
<td>1.65</td>
</tr>
<tr>
<td>64</td>
<td>3.81</td>
<td>3.04</td>
</tr>
<tr>
<td>128</td>
<td>7.65</td>
<td>6.33</td>
</tr>
<tr>
<td>256</td>
<td>15.06</td>
<td>12.96</td>
</tr>
<tr>
<td>512</td>
<td>30.40</td>
<td>12.99</td>
</tr>
<tr>
<td>1024</td>
<td>59.67</td>
<td></td>
</tr>
</tbody>
</table>

Execution rates in Gflops/s.