CMS: An Integrated Simulation Environment

J. Leslie Walker and Abbas Birjandi

College of Computer Science, Northeastern University, Boston Massachusetts

Abstract

The Connection Machine is a massively parallel architecture designed for general computation. This paper describes a simulation environment which allows one to execute programs written for a connection machine. The CMS simulation environment provides several tools for empirical analysis of these parallel algorithms. First, it provides performance statistics on the simulated connection machine as it runs different user programs. These statistics compiled from the simulated execution of a set of algorithms provide a measure by which one may arrive at the optimal solution from that set. Secondly, The behavior of the simulated environment is governed by configuration parameters which may be altered to find the optimal connection machine configuration in which to run an algorithm. The CMS simulation environment runs on the Sun Workstation and uses the windowing and mouse interfaces provided by the SunView™ integrated application environment. Different windows allow the user to view different parts of the simulated connection machine at the same time. This paper also includes a discussion of the internals of the simulator and the reasoning behind the methods used. By examining these details one can learn more about the hardware implementation of the connection machine.

Overview

This paper provides an introduction to CMS, a simulator for a connection machine [Hillis 86]. The goal of the simulation is to provide an environment in which to run connection machine algorithms and gather detailed statistics about their execution. The simulator runs in an integrated environment which provides windowing and support for a mouse. The simulator software and integrated display simplifies the presentation of the connection machine hardware enough to make it understandable to the novice user while keeping it accurate enough to represent the connection machine hardware.

The connection machine is a massively parallel computer architecture. The granularity of the processors which compose a connection machine is very small—one bit. The architecture works most efficiently on large, uniform problems with very small granularity. Some examples are database query searches and kernel convolutions.

The algorithms that run on a connection machine need not be vectorized to run efficiently. In fact, the processors may be considered to be arranged with complete adjacency, meaning that every processor is directly adjacent to every other processor. This means that trees, graphs, and other geometries may be implemented on the connection machine as well as vectors.

Connection Machine Architecture

Interfacing

The connection machine is not equipped to perform system tasks such as device control and data acquisition. For this reason it must interface to a host computer which provides these types of services. The host has full access to each processor's memory and also provides the instruction stream which the individual processors will execute. The host usually will provide an initial state for the connection machine by depositing the appropriate data into the memory of each of the processors. The host may also retrieve results from the processors' memories during execution or afterwards.

Cells

Processors in the Connection Machine are referred to as processing cells. Each processing cell is able to independently execute the basic building block operations of the whole machine. These processing cells are identical and each contains its own memory, registers, and ALU. Each of these components as well as all of the
corresponding data paths are uniformly one bit wide. The single-bit registers in a cell are referred to as cell flags. General Purpose cell flags may be set, cleared, or read by the cell without side effects. These are used to store a single bit of information. Special cell flags have side effects to being set, cleared, or read. These flags are the means by which a cell communicates with other cells and the routers.

The connection machine is a SIMD (Single-Instruction-Multiple-Data) multiple processor architecture. The instruction is broadcast to each cell via the global instruction bus.

The Network

Local Areas The processing cells in the connection machine are grouped into uniform sets which we shall call local areas. The cells in a local area share several communication structures:

- **daisy chain** The output of each cell’s ALU is linked with a special purpose register on another cell in such a way that a circular chain is formed. Data may be passed through this chain in serial fashion.

- **NEWS** Similarly to the daisy chain, the output of each cell’s ALU is linked with a special purpose register in four other cells so that the cells form a square with each cell able to communicate directly with cells in four directions.

- **router** A special purpose register of each cell in a local area is linked with the router for that local area. The routers form a network that establishes the complete adjacency of the architecture.

In this simulation cells are referred to in relation to their local area. Each cell has a number in it’s local area and that local area has a router number. So, each cell is referred to as:

<cell,router>

Network Topology It is obvious that with groups of cells being served by routers, all cells are not actually directly adjacent. Complete adjacency is established in the architecture by the fact that each cell can communicate indirectly with any other cell in the connection machine without cells being a part of the communication process except at the terminal points. In other words cells only see messages that they send or are sent to them – a cell will never receive a message that it must pass on. The task of transmitting messages is handled by the network. The network is composed of routers and data paths. Routers are described below. The data paths are bi-directional links between routers.

The routers are connected to each other by bi-directional data paths which allow routers to transmit messages to each other in both directions. The adjacencies made by these data paths form a boolean n-cube in which no router is further than \( \lg n \) data paths from any other router. Messages are addressed to a particular cell within the local area of a particular router. The cell is addressed by a cell number \((\text{cell address})\), while the router is referred to by it’s location on the n-cube relative to the router that is sending the message \((\text{router address})\). The router address contains all the information necessary to route the message at any point in it’s transmission.

Each router has \( \lg n \) data paths which establish it’s adjacency with \( \lg n \) other routers on the network. These data paths may be enumerated so that path \( i \) on any given router is known as path \( i \) to the router on the other end. In this way if two routers both transmit a message along path \( i \) they exchange messages with each other. The numbers that are given to the paths are called the path’s dimension and all the routers in the network transmit on only one dimension at a time.

When all of the routers transmit on one dimension it is referred to as a dimension cycle. The network executes dimension cycles for each dimension consecutively beginning with the first:

\[
1 \ 2 \ \ldots \ \lg n
\]

Once the network has executed a dimension cycle for all \( \lg n \) dimensions it starts over again at the first dimension. \( \lg n \) dimension cycles make up one petite cycle in which dimension cycles are executed for all of the dimensions:

\[
\begin{align*}
1 \\
2 \\
\vdots \\
\lg n
\end{align*} = \text{petite cycle}
\]

Routers The routers are composed mainly of buffers and termination points for data paths. Since the data paths are of limited capacity, the router saves it’s messages in buffers. Messages have priorities and addresses that regulate their transmission. Addresses tell which data path a message needs to travel on and priorities are used to resolve the collision that occurs when more than one buffered message needs to travel on a single message path at the same time.

The Simulator

We have constructed a piece of software that simulates the connection machine described above. This software simulates the hardware from the host machine interface down to the level of each bit of the connection machine. Cells, routers, and data paths are all visually represented.
The simulator runs as a separate process that receives instructions and memory requests from a user application. The user application starts the simulator and establishes several lines of communication with it. These communication lines are analogous to the interface with the host machine and the user application is analogous to the process running on that host machine. The communication lines allow instructions to be sent to the simulator and allow access to the cell memory one byte at a time. The disk DMA path is not supported.

The network is composed of 256 routers each having 16 local processing cells. Each cell has 2k bits of local memory.

Operating Environment

CMS is implemented under Unix using the SUNView integrated application environment. All displays that are called control panels are implemented using the SUNView panel objects and SUNView canvases. It is sufficient for the user to familiarize themselves with the operation of these objects to use most of the mouse commands in the simulator.

User Interface

The simulator provides an interface through SUNView which behaves much like a traditional machine level debugger.

Windows

The simulator visually represents the components of the simulated connection machine through several windows. These windows may be overlapped in the SUNView environment as shown in Figure 2. There are separate windows for:

- Cells
- Routers
- Complete Network
- Global Instruction Bus

These windows contain control panels which provide access to the commands via the mouse. The data in the windows is updated in real-time. For example, when a

![Figure 2: A sample CMS screen](image-url)
memory request comes through from the host to store a value in memory and the cell which owns that memory is shown, then the window which shows that cell will be updated with the new memory value.

Commands

The simulator commands are meant to allow it to function similarly to a traditional machine language debugger. There are commands to alter the contents of cell memory and cell flags, execute instructions, and even alter the behavior of the machine. All commands may be entered as text in a command line format, and some are accessible via pushbuttons and control panels. There is also an indirect command file capability for automated execution.

The area of the windows which displays the data is active in the sense that clicking the mouse over certain areas of the display will cause commands to be executed. Some examples:

- Clicking the mouse over a cell flag will cause the flag to change in value to a '0' or '1'.
- Clicking the mouse over one of the adjacencies in the router window will cause the window to switch from displaying the old router to display the router adjacent to it on the dimension that was selected.

Application Interface

In the actual connection machine hardware, a conventional computer is required to host the connection machine hardware. The host provides a front end for the connection machine’s specialized computing power. It is essential for the simulator to have this design also so that the programmer has a feel for the abstraction provided by the interface between the connection machine and the host. The connection between the host and the machine is called the microcontroller.

The microcontroller functions as both an interface to the connection machine hardware, and a low-level abstraction of the connection machine’s functionality. In general one macro-instruction is translated by the microcontroller into several connection machine instructions called nano-instructions. Because one macro-instruction can produce many nano-instructions the microcontroller must buffer the nano-instructions in a FIFO buffer.

The simulator provides a microcontroller as well. This microcontroller passes nano-instructions which are produced by the application through a pipe to the simulator which will read them in one at a time and execute them. The simulator in turn responds through another pipe for each instruction, thus providing the necessary synchronization. The microcontroller is divided into 2

![Diagram of relationship between microcontrollers.](image)

modules: the application microcontroller (used by the host) and the simulator microcontroller (used by the simulator).

The Application Microcontroller

The application microcontroller provides access to the connection machine from the user application program. The goal in its design was for the application to appear to be talking to an actual hardware device whose memory was attached to the bus. However, to be realistic the microcontroller must also provide macro-instructions and translate these into nano-instructions for the machine to execute. The decision was made to do this on the application side so that the user is able to experiment with his own high-level abstractions of the connection machine operations.

The Simulator Microcontroller

The microcontroller on the simulator is not under the same burden of having to appear to be talking to hardware. It is implemented as a separate module of the simulator software. There is no direct communication between the programmer and the simulator microcontroller, except via the application microcontroller.

References


[Think 87-4] *Connection Machine Model CM*-