The Connection Machine

W. Daniel Hillis

The MIT Press
Cambridge, Massachusetts
London, England

This book was set using LaTeX by Thinking Machines Corporation and printed and bound by Halliday Lithograph in the United States of America. The terms in the index were selected automatically using The TMC Indexer.

Library of Congress Cataloging-in-Publication Data

Hillis, W. Daniel.
The connection machine.

(The MIT Press series in artifical intelligence)
Thesis (Ph.D.)—MIT, 1985.
Bibliography: p.
Includes index.
1. Machine theory. 2. Artificial intelligence.
I. Title. II. Series.
QA267.H487 1985 004 85–15318
ISBN 0–262–08157–1

SERIES FOREWORD

Artificial intelligence is the study of intelligence using the ideas and methods of computation. Unfortunately, a definition of intelligence seems impossible at the moment because intelligence appears to be an amalgam of so many information-processing and information-representation abilities.

Of course psychology, philosophy, linguistics, and related disciplines offer various perspectives and methodologies for studying intelligence. For the most part, however, the theories proposed in these fields are too incomplete and too vaguely stated to be realized in computational terms. Something more is needed, even though valuable ideas, relationships, and constraints can be gleaned from traditional studies of what are, after all, impressive existence proofs that intelligence is in fact possible.

Artificial intelligence offers a new perspective and a new methodology. Its central goal is to make computers intelligent, both to make them more useful and to understand the principles that make intelligence possible. That intelligent computers will be extremely useful is obvious. The more profound point is that artificial intelligence aims to understand intelligence using the ideas and methods of computation, thus offering a radically new and different basis for theory formation. Most of the people doing artificial intelligence believe that these theories will apply to any intelligent information processor, whether biological or solid state.

There are side effects that deserve attention, too. Any program that will successfully model even a small part of intelligence will be inherently massive and complex. Consequently, artificial intelligence continually confronts the limits of computer science technology. The problems encountered have been hard enough and interesting enough to seduce artificial intelligence people into working on them with enthusiasm. It is natural, then, that there has been a steady flow of ideas from artificial intelligence to computer science, and the flow shows no sign of abating.

The purpose of this MIT Press Series in Artificial Intelligence is to provide people in many areas, both professionals and students, with timely, detailed information about what is happening on the frontiers in research centers all over the world.

Patrick Henry Winston Michael Brady

ACKNOWLEDGMENTS

In alphabetical order:

Hal Abelson, who encouraged me to work on this in the early stages. Phil Agre, who did some of the first programming on the machine. Jim Bailey, who appreciated the elegance from the first time he heard about it and explained it to everyone else. Alan Bawden, who invented the first programming languages for the machine. Gordon Bell, who gave lots of advice that I listened to and some that I wish I had. Danny Bobrow, who took me seriously. Michael Brady, for encouragement. Keira Bromberg, who was relentless. Tom Callahan, who built it; nothing is impossible for him. David Chapman, who had the first ideas about how to make conventional programs run on the machine. Dave Christman, who invented many of the basic algorithms. Arlene Chung, for making the pictures prettier. Dick Clayton, who made it real. It never would have happened without him, really. John Cocke, for discussions on architecture. Jim Cohen, for constant support. Glen Kramer, who figured out how to use the machine for test vector generation. Marvin Denicoff, for understanding. Michael Dertouzos, who gave me the support to build my first parallel processor. Chris Drake, for good humor. Gary Drescher, for teaching me to think better. Mike Drumheller, who wrote the first vision program. Scott Fahlman, whose thesis inspired the machine. Carl Feynman, for energy, joy, and enthusiasm from the beginning. Richard Feynman, for teaching me something about what is important and what is not. Rolf-Dieter Fiebrich, who led the group that wrote the first useful programs for the machine, with a sense of wonder and excitement. Craig Fields, who knew it was the right thing. Richard Greenblatt, for advice and encouragement. Sheryl Handler, for unending confidence in me and for showing me the power and perils of positive thought. Robert Heinlein, for making me want to go to MIT. Carl Hewitt, for good discussions. John Huffman, who designed the chip. Lyman Hurd, who analyzed the router. Brewster Kahle, one of the machine's primary designers, for boundless effort, friendship and excitement, which makes it all worthwhile. Bob Kahn, who supported the machine throughout its development. John Kimberly, who designed the I/O for the prototype. Tom Knight, who was one of the principal designers of the first prototype. Bradley Kuszmaul, who figured out how to compile functional programs for the machine. Cliff Lasser, who wrote the first language that worked on the machine. Jerry Letvin, for talking me out of being a neurophysiologist. Clem Liu, who was one of the principal designers of the prototype. Pati Marx, for love and support. Antonio Marzullo, for being there when I needed him. Boris Mashalov, for keeping things apart. Neil Mayle, who wrote the initial simulations. Mirza Mehdi, for making the deals. Margaret Minsky, who helped and encouraged me, for good criticism, ideas and companionship. Marvin Minsky, my mentor, who taught me to think. Most of the ideas in the book have their root, directly or indirectly, in discussions with Marvin. (See also Thesis Committee.) The Minsky family, Julie, Henry, Margaret, Gloria and Marvin, who adopted me when I moved to Boston. Guy Montpetit for joy and his sense of the adventure. Paul Mott, for indexing on the Fourth of July. Bruce Nemnich, who made things work. William Paley, for believing in the dream. Seymour Papert, for supervising my first projects at MIT. Tom Poggio, who is my model of a true scientist. Michael Rabin, for discussions on algorithms. Howard Resnikoff, for finding order in chaos (and Sheryl Handler for the reverse). George Robertson, who wrote the first code to run on the hardware. Paul Rosenblum, who kept up the dress code. Jim Salem, who accidently wrote the router diagnostic. Jack Schwartz, who took the trouble to understand the details. Claude Shannon, whose playful spirit got the field off to a good start. (See also Thesis Committee.) Brian Silverman, for early discussions on the router. Karl Sims, who is the first person to apply the machine to graphics. Mark Stefik, for early encouragement. Steve Squires, for support, ideas, and enthusiasm. Frank Stanton, for advice and support. Guy Steele, who headed the software team for the prototype. Dave Stefanovic, who made things work. Gerald Sussman, who gave me good advice and encouragement. Many of the key early ideas came out of discussions with Gerry. (See also Thesis Committee.) Ivan Sutherland, who said to think about wires instead of switches. John Taft, who was one of the designers of the first prototype. Wati Taylor, who figured out how to do sorting. Tamiko Thiel, who made it beautiful. Umesh Vasirani, who wrote one of the initial simulations. Dave Waltz, who is applying the machine to natural language understanding. Dan Weinreb, who wrote one of the initial simulations. Debbie Widener, for writing the book. Jerry Wiesner, for encouragement and advice. Patrick Winston, who supported the work from the beginning. (See also Thesis Committee.) Steve Wolfram, for friendship, ideas, and encourgement. The people and creatures of Disney World, who guarded me while I wrote the first draft. Lowell Wood, who recognized my potential before I did but fortunately was unable to exploit it, for encouragement, humor and advice. I would also like to thank my Thesis Committee, Marvin Minsky (thesis advisor), Claude Shannon (advisor), Gerald Sussman (thesis advisor), Patrick Winston; the Defense Advanced Research Projects Agency and the Naval Electronic Systems Command for support of the construction of the prototype under contract #N00039-84-C-0638; the Fannie and John Hertz Foundation, which supported me for six long years without complaint; and the editorial staff of The MIT Press.

The book is dedicated to my family, Beth, David, Argye and Bill, for years of love and support.

The Connection Machine

Contents

Chapter 1

Introduction

1.1 We Would Like to Make a Thinking Machine

Someday, perhaps soon, we will build a machine that will be able to perform the functions of a human mind, a thinking machine. One of the many problems that must be faced in designing such a machine is the need to process large amounts of information rapidly, more rapidly than is ever likely to be possible with a conventional computer. In this book I describe a new type of computing engine called a *Connection Machine*; it computes through the interaction of many, say a million, simple identical processing/memory cells. Because the processing takes place concurrently, the Connection Machine Computer can be much faster than a traditional computer.

Our Current Machines Are Too Slow

Although the construction of an artificial intelligence is not yet within our reach, the ways in which current computer architectures fall short of the task are already evident. Consider a specific problem. Let us say that we are asked to describe, in a single sentence, the picture shown in figure 1.1. With almost no apparent difficulty a person is able to say something like "It is a group of people and horses." This is easy for us. We do it almost effortlessly. Yet for a modern digital computer it is an almost impossible task. Given such an image, the computer first has to process the hundreds of thousands of points of visual information in the picture to find the lines, the connected regions, and the textures of the shadows. From these lines and regions it then constructs some sort of three-dimensional model of the shapes of the objects and their locations in space. Then it has to match these objects against a library of known forms to recognize the faces, the hands, the folds of the hills, etc. Even this is not sufficient to make sense of the picture. Understanding the image requires a great deal of commonsense knowledge about the world. For example, to recognize the simple waving

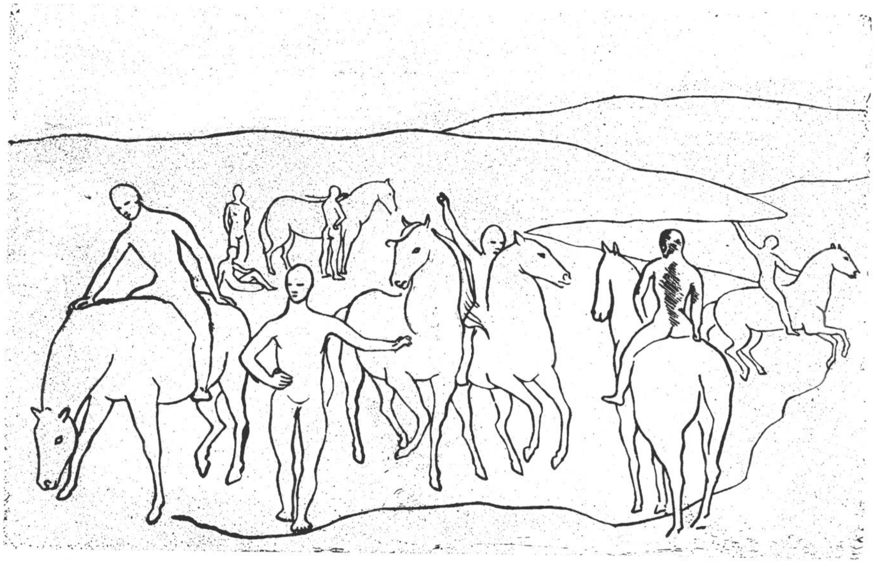

Figure 1.1 *The Watering Place*, Pablo Picasso, 1905.

lines as hills, one needs to expect hills; to recognize horses' tails, one needs to expect a tail at the end of a horse.

Even if the machine had this information stored in its memory, it would probably not find it without first considering and rejecting many other possibly relevant pieces of information, such as that people often sit on chairs, that horses can wear saddles, and that Picasso sometimes shows scenes from multiple perspectives. As it turns out, these facts are all irrelevant for the interpretation of this particular image, but the computer would have no a priori method of rejecting their relevance without considering them. Once the objects of the picture are recognized, the computer then has to formulate a sentence which offers a concise description. This involves understanding which details are interesting and relevant and choosing a relevant point of view. For example, it probably would not be satisfactory to describe the picture as "Two hills, partially obscured by lifeforms," even though this may be accurate.

We know just enough about each of these tasks so that we might plausibly undertake to program a computer to generate one-sentence descriptions of simple pictures, but the process would be tedious, and the resulting program would be extremely slow. What the human mind does almost effortlessly would take the fastest existing computers many days. These electronic giants that so outmatch us in adding columns of numbers are equally outmatched by us in the processes of symbolic thought.

The Computer versus the Brain

So what's wrong with the computer? Part of the problem is that we do not yet fully understand the algorithms of thinking. But part of the problem is speed. One might suspect that the reason the computer is slow is that its electronic components are much slower than the biological components of the brain, but this is not the case. A transistor can switch in a few nanoseconds, about a million times faster than the millisecond switching time of a neuron. A more plausible argument is that the brain has more neurons than the computer has transistors, but even this fails to explain the disparity in speed. As near as we can tell, the human brain has about 10^{10} neurons, each capable of switching no more than a thousand times a second. So the brain should be capable of about 10^{13} switching events per second. A modern digital computer, by contrast, may have as many as 10^9 transistors, each capable of switching as often as 10^9 times per second. So the total switching speed should be as high as 10^{18} events per seconds, or 10,000 times greater than the brain. Thus the sheer computational power of the computer should be much greater than that of the human. Yet we know the reality to be just the reverse. Where did the calculation go wrong?

1.2 Classical Computer Architecture Reflects Obsolete Assumptions

One reason that computers are slow is that their hardware is used extremely inefficiently. The actual number of events per second in a large computer today is less than one-tenth of one percent of the number calculated in section

1.1. The reasons for the inefficiency are partly technical but mostly historical. The basic forms of today's architectures were developed under a different set of technologies, when different assumptions from those that are appropriate today were applied. The machine described here, the Connection Machine, is an architecture that better fits today's technology and, I hope, better fits the requirements of a thinking machine.

A modern large computer contains about 1 m^2 of silicon. This square meter contains approximately one billion transistors which make up the processor and memory of the computer. The interesting point here is that both the processor and memory are made of the same stuff. This was not always the case. When von Neumann and his colleagues were designing the first computers, their processors were made of relatively fast and expensive switching components, such as vacuum tubes, whereas the memories were made of relatively slow and inexpensive components, such as delay lines or storage tubes. The result was a two-part design that kept the expensive vacuum tubes as busy as possible. We call this two-part design, with memory on one side and processing on the other, the von Neumann architecture, and it is the way that almost all computers are built today. This basic design has been so successful that most computer designers have kept it even though the technological reason for the memory/processor split no longer is justified.

The Memory/Processor Split Leads to Inefficiency

In a large von Neumann computer almost none of its billion or so transistors do any useful processing at any given instant. Almost all the transistors are in the memory section of the machine, and only a few of those memory locations are accessed at any given time. The two-part architecture keeps the silicon devoted to processing wonderfully busy, but this is only 2 or 3 percent of the silicon area. The other 97 percent sits idle. At a million dollars per square meter for processed, packaged silicon, this is an expensive resource to waste. If we were to take another measure of cost in the computer, kilometers of wire, the results would be much the same: Most of the hardware is in memory, so most of the hardware is doing nothing most of the time.

As we build larger computers, the problem becomes even worse. It is relatively straightforward to increase the size of memory in a machine, but it is far from obvious how to increase the size of the processor. The result is that as we build bigger machines with more silicon, or, equivalently, as we squeeze more transistors into each unit of area, the machines have a larger ratio of memory to processing power and are consequently even less efficient. This inefficiency remains no matter how fast we make the processor because the length of the computation becomes dominated by the time required to move data between processor and memory. This is called the von Neumann bottleneck. The bigger we build machines, the worse it gets.

1.3 Concurrency Offers a Solution

The obvious answer is to get rid of the von Neumann architecture and build a more homogeneous computing machine in which memory and processing are combined. It is not difficult today to build a machine with hundreds of thousands or even millions of tiny processing cells which has a raw computational power that is many orders of magnitude greater than the fastest conventional machines. The problem lies in how to couple the raw power with the applications of interest, how to program the hardware to the job. How do we decompose our application into hundreds of thousands of parts that can be executed concurrently? How do we coordinate the activities of a million processing elements to accomplish a single task? The Connection Machine architecture was designed as an answer to these questions.

Why do we even believe that it is possible to perform these calculations with such a high degree of concurrency? There are two reasons. First, we have the existence proof of the human brain, which manages to achieve the performance we are after with a large number of apparently slow switching components. Second, we have many specific examples in which particular computations can be achieved with high degrees of concurrency by arranging the processing elements to match the natural structure of the data.

Image Processing: One Processor per Pixel

In image processing, for example, we know that it is possible to perform two-dimensional filtering operations efficiently using a two-dimensionally connected grid of processing elements. In this application it is most natural to store each point of the image in its own processing cell. A 1000×1000 point image would use a million processors. In this case, each step of the calcu-

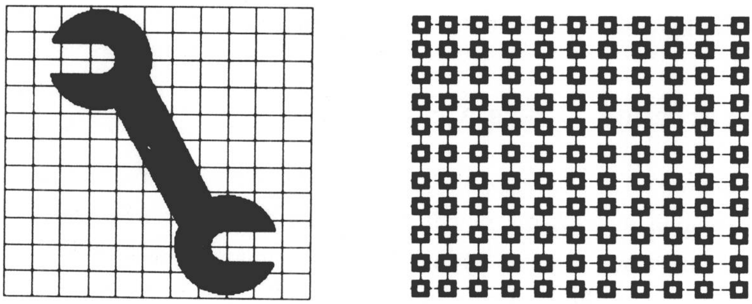

Figure 1.2 In a machine vision application, a separate processor/memory cell processes each point in the image. Because the computation is two-dimensional the processors are connected into a two-dimensional grid.

lation can be performed locally within a pixel's processor or through direct communication with the processors' two-dimensionally connected neighbors. (See figure 1.2.) A typical step of such a computation involves calculating for each point the average value of the points in the immediate neighborhood. Such averages can be computed simultaneously for all points in the image. For instance, to compute the average of each point's four immediate neighbors requires four concurrent processing steps during which each cell passes a value to the right, left, below, and above. On each of these steps the cell also receives a value from the opposite direction and adds it to its accumulated average. Four million arithmetic operations are computed in the time normally required for four.

VLSI Simulation: One Processor per Transistor

The image processing example works because the structure of the problem matches the communication structure of the cells. The application is two dimensional, the hardware is two dimensional. In other applications the nat-