DESIGN SKETCH FOR A MILLION-ELEMENT NETL MACHINE

Scott E. Fahlman

Carnegie-Mellon University
Department of Computer Science
Pittsburgh, Pennsylvania 15213

Abstract

This paper describes (very briefly) a parallel hardware implementation for NETL-type semantic network memories. A million-element system can be built with about 7000 IC chips, including 4000 64K RAM chips. This compares favorably with the hardware cost of holding the same body of knowledge in a standard computer memory, and offers significant advantages in flexibility of access and the speed of performing certain searches and deductions.

1. Introduction

In [1] I presented a scheme for representing real-world knowledge in the form of a hardware semantic network. In this scheme, called NETL, each node and link in the network is a very simple hardware processing element capable of passing single-bit markers through the network in parallel. This marker-passing is under the overall control of an external serial processor. By exploiting the parallelism of this marker-passing operation, we can perform searches, set intersections, inheritance of properties and descriptions, multiple-context operations, and certain other important operations much faster than is possible on a serial machine. These new abilities make it possible to dispense with hand-crafted search-guiding heuristics for each domain and with many of the other procedural attachments found in the standard AI approaches to representing knowledge. In addition to the difficulty of creating such procedures, and the very great difficulty of getting the machine to create them automatically, I argue that the heuristic systems are brittle because they gain their efficiency by ignoring much of the search space. NETL, on the other hand, looks at every piece of information that might be relevant to the problem at hand and can afford to do so because it does not have to look at each piece of information serially.

NETL has been viewed by many in the AI community as an interesting metaphor and a promising direction for future research, but not as a practical solution to current AI problems because of the apparently impossible cost of implementing a large NETL system with current technology. The problem is not that the hardware for the nodes and links is too costly -- hundreds or even thousands of these elements can be packed onto a single VLSI chip. Rather, the problem is in forming new private-line connections (wires) between particular nodes and links as new information is added to the network. These connections cannot be implemented as signals on a single shared bus, since then all of the parallelism would be lost. Indeed, it is in the pattern of connecting wires, and not in the hardware nodes and links, that most of the information in the semantic network memory resides. A large switching network, similar to the telephone switching network, can be used in place of physical wires, but for a network of a million elements one would need the functional equivalent of a crossbar switch with 4×10^{12} crosspoints. Such a switch would be impossibly expensive to build by conventional means.

In the past year I have developed a multi-level time-shared organization for switching networks which makes it possible to implement large NETL systems very cheaply. This interconnection scheme, which I call a **hashnet** because some of its internal connections are wired up in a random pattern, has many possible uses in non-AI applications; it is described in its general form in another paper [2]. In this paper I will briefly describe a preliminary design, based on the hashnet scheme, for a semantic network memory with 10^6 NETL elements. (An "element" in NETL is the combination of a single node and a four-wire link.) A million-element NETL system is 10-20 times larger than the largest AI knowledge bases in current use, and it offers substantial advantages in speed and flexibility of access. It is an open question whether a knowledge-base of this size will be adequate for common-sense story understanding, but a system of 10^6 NETL elements should hold enough knowledge for substantial expertise in a variety of more specialized domains. In a paper of this length I will be able to sketch only the broad outlines of the design -- for a more complete account see [3].

The NETL machine itself, excluding the serial control computer, requires about 7000 VLSI chips, 4000 of which are commercial 64K dynamic RAM chips. (See the parts list, table 1.) As we will see later, with the same 64K memory technology it would require a comparable number of chips to store the same body of information in a conventional Planner-style data base, assuming that the entire data base is kept in a computer's main memory and not on disk. So, far from being impossibly expensive, this scheme is quite competititve with standard random-access memory organizations. I am about to seek funding to build a million-element prototype machine within the next two or three years. The 64K RAM chips are not available today in sufficient quantities, and may be quite expensive for the next couple of years. The prototype machine will be designed so that 16K RAMs can be substituted if necessary, giving us a 256K element machine to use until the 64K RAM chips are obtained.

2. Requirements of NETL

Figure 1 shows a basic NETL element as it was originally conceived. Commands from the control computer are received over the common party-line bus. The applicability of any command to a given element depends on the element's unique serial number, on the state of 16 write-once **flag bits** which indicate what type of node or link the element represents, and on

[1]This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-78-C-1551. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

the state of 16 read-write **marker bits** which indicate the current state of the element. These marker bits represent the short-term memory in the system. Also present are some number (4 in this design) of distinct **link wires**, and a **node terminal** to which link wires from other elements can be connected. Commands typically specify that all elements with a certain bit-pattern should send a one-bit signal across incoming or outgoing link wires, and that any element receiving such a signal should set or clear certain marker bits. It is also possible to address a command to a specific element, or to get any element with a certain marker pattern to report its serial number over the common bus. Using these commands, it is possible to propagate markers through the network Quillian-style or to control the marker propagation in any number of more precise ways. For details, see [1], especially section 2.3 and appendix A.1.

In a million-element design, then, we have 4 sets of 10^6 link wires to connect to 10^6 node terminals, each by a private line. A link wire is connected to only one node terminal, but a node terminal may have any number of link wires attached to it. Unlike the telephone system, this network must support all 4 million connections simultaneously; once a connection is made, it becomes part of the system's long-term memory, and a connection is seldom, if ever, released. As the system learns new information, new links are wired up one at a time, and this must be done without disturbing the connections already in use. If the same network hardware is to be used for different knowledge bases at different times, it must be possible to drop one set of connections and replace them with a new set.

A few additional constraints will help us to separate interesting designs from uninteresting ones. If we want to obtain a roughly human-like level of performance in our knowledge base, the transmission of a bit from one element to another can take as long as a few milliseconds. Since answering a simple question -- for example, determining whether an elephant can also be a cabbage -- takes something like 20 to 60 basic marker-propagation cycles, a propagation time of 5 milliseconds gives a response time of .1 to .3 seconds. This figure is independent of the total size of the network. This means that some degree of parallelism is essential, but that with microsecond-speed technologies there is room for some time-sharing of the hardware as well.

New connections are established individually, and setting them up can take somewhat more time than simple propagations: humans are able to add only a few items to long-term memory per second. If an attempt to create a new connection should fail occasionally, nothing disastrous occurs -- the system simply skips over the link it is trying to wire up and goes on to the next free one.

3. The NETL Machine Design

As mentioned earlier, the key problem here is to build a switching network for connecting link-wires to nodes. Since the four wires of a link are used at different times, we can think of this switch as four separate sub-netowrks, each with 10^6 inputs, 10^6 outputs, and with all 10^6 connections operating at once. This network must be, in the jargon of network theory, a **seldom-blocking network**. It must be possible to add new connections one by one, without disturbing any connections that are already present, but some small chance of failure in establishing new connections can be tolerated. Once established, a connection must work reliably, and must be able to transmit one-bit signals in either direction. Note that by "connection" here I mean a setting of the network switches that establishes a path from a given input to a given output; the pysical wiring of the network is of course not altered during use.

The basic concept used in this design is to build a 960 x 960 seldom-blocking switching network, then to time-share this network 1024 ways. The number 960 arises from packaging considerations; this gives us a total of 983,040 virtual connections, close enough to one million for our purposes. The 1024 time slices roll by in a regular cycle; a different set of switch settings is used during each slice. There are four sets of 1024 switch settings, corresponding to the four link-wire sub-netowrks. The bits describing the 4096 settings for each switch are stored in random access memory chips between uses. The NETL elements are likewise implemented in a time-shared fashion: 960 element units are implemented in hardware (four to a chip with shared bus decoders), and each of these element devices is time shared 1024 ways. Each NETL element exists in hardware only during its assigned time-slice; most of the time, it exists only as 32 bits of state in a memory chip.

Let us begin by considering the construction of a 960 x 960 hashnet without time-sharing. The basic unit of construction is the 15-way selector cell shown in figure 2a. This cell connects its input to any of its 15 outputs, according to the contents of a four-bit state register. A value of 0 indicates that the cell is currently unused and that its input is not connected to any output. A single integrated circuit chip can easily hold 15 of these selector cells; the corresponding outputs from each cell are wired together internally, as shown in figure 2b. With assorted power and control lines, this 15 x 15 switching element requires a 48-pin package.

To build a 960 x 960 seldom-blocking network out of these elements, we arrange them in four layers with 1920 selector cells (128 chips) in each. (See figure 3.) The outputs of each layer are wired to the inputs of the next layer with a fixed but randomly chosen pattern of wires. Each of the input terminals of the hashnet is wired to 2 selector cells in the first layer; each of the outputs of the hashnet is wired to 2 outputs lines from the last layer of cells. Initially all of the selector cells are in the non-busy state. As paths through the network are set up, each one uses up one of the selector cells in each layer. Note, however, that half of the selector cells remain unused even when all 960 connections are in place; this spare capacity ensures that the network will have a low chance of blocking even for the last few connections.

To set up a new connection from a given input to a given output, we first broadcast a marking signal through the network from the input to all of the selector cells and outputs that can be reached. Only non-busy cells play a role in this process. If this signal reaches the desired output, one of the marked paths is traced back toward the source, with the selector cells along the way being set up appropriately. These cells become busy and will not participate in any other connections. Since the inter-layer wiring of the network is random, and since we are using many fewer switches than are needed for a strictly non-blocking network, we cannot guarantee that a desired connection can be found. We can guarantee that the probability of being unable to find a desired connection is very small. In simulation tests of this design, 100 complete sets of connections were attempted -- 10,000 connections in all -- with only 2 failures. As noted earlier, an occasional failure to find a connection is not disastrous in the NETL application; we just try again with a different link.

Instead of using four layers of selector cells, it is possible to get the same effect using only one layer. The output wires of this layer are randomly shuffled and fed back to the inputs; these wires also go to the network's ouput terminals. Four sets of switch settings are used, corresponding to the four layers of the original network. The signal bits are read into this layer of cells and latched. Using the first-layer switch settings, they are sent out over the appropriate output wires and shuffled back to the inputs where these bits are latched again. Then the second layer setup is used and the signal bits are shuffled again. After four such shuffles, the bits are in the right place and are read from the network's outputs. We have traded more time for fewer switches and wires; the same number of setup bits are used in either case.

To go from a thousand connections to a million, we time-share this network 1024 ways. This requires two modifications to the network. First, instead of having only 4 bits of state for each selector cell (or 16 bits if the shuffling scheme is in use), we need a

different set of bits for each time slice. These bits are read from external memory chips; 4 bits of setup are read in through each selector cell input, followed by the signal bit. Second, since the NETL elements are permanently tied to their assigned time-slices, we need some way to move the signal bits from one time-slice to another. This operation is carried out by time-slice shifter chips. Each of these devices is essentially a 1024-bit shift register. During one cycle of time-slices this shift register is loaded: during each slice, a signal bit is received along with a 10-bit address indicating the slice that the signal bit is to go out on. The address governs where in the shift register the signal bit is loaded. On the next cycle of slices, the bits are shifted out in their new order. An entire layer of 1920 time-shifters is needed (packed 5 to a chip), along with memory chips to hold the 10-bit time-slice addresses. The chance of blocking is minimized if these are placed in the center of the network, between the second and third layers of cells. Some additional chance of blocking is introduced by addition of the shifters to the network, but not too much. In our simulations of an almost-full network with time sharing, we encountered 37 blocked connections in 110,000 attempts.

4. Cost and Performance

As can be seen from the parts list, most of the cost of the NETL machine is in the memory chips. In order to keep the number of chips below 10,000 -- a larger machine would be very hard to build and maintain in a university research environment -- 64K dynamic RAM chips have been used wherever possible in the design. The memory associated with the element chips must be either 2K x 8 static RAMS or fast 16K dynamic RAMS for timing reasons. In fact, the limited bit-transfer rate of the memory chips is the principal factor limiting the speed of the network; if 16K x 4 chips were available, the system could be redesigned to run four times as fast.

As it currently stands, the system has a basic propagation time of about 5 milliseconds. This is the time required to accept 10^6 bits and to steer each of these to its independent destination. This assumes the use of 2K x 8 static RAMs for the element memories and allows for a rather conservative page-mode access time of 200 nanoseconds for the 64K RAM chips. (In page mode, half of the address bits remain constant from one reference to the next.) The 256K element version, using 16K RAM chips, should have a propagation time of 1.25 milliseconds.

Since the million-element machine contains 4000 64K RAM chips, the parts cost of the machine is tied closely to the price of these chips. Currently, if you can get 64K RAMs at all, they cost over $100, but the price is likely to drop rapidly in the next two years. If we assume a price of $20 for the 64K RAMs and $10 for the 2Kx8 RAMs, we get a total of roughly $100,000 for the memory chips in the machine. It is also hard to know what price to assign to the three custom chips in the design. An initial layout cost of $150,000 for all three would probably be reasonable, but this cost would occur only once, and the chips themselves should be easy to fabricate. We have not yet thought hard about board-level layout, but rough calculations suggest that the machine would fit onto about 52 super-hex boards of two types. Two standard equipment racks ought to be sufficient for the whole machine.

For comparison, it might be worthwhile to calculate the cost of storing the same information in a Planner-style data base in the memory of a serial machine. To make the comparison fair, let us assume that the entire data base is to be kept in main memory, and not paged out onto disk. To store the information in the network itself, in the simplest possible form, would require 80 million bits of memory: 4 million pointers of 20 bits each. This would require 1250 64K RAM chips. A simple table of pointers, of course, would be very slow to use. If we add back-pointers, the figure doubles. If we add even the most limited sort of indexing structure, or store the entries in a hash table or linked list, the amount of memory doubles again. At this point, we have reached 4000 64K RAM chips, the same number used in the NETL machine. The moral of the story would seem to be that the greater power and flexibility of the NETL organization can be had at little, if any, extra cost.

Acknowledgements

Ed Frank and Hank Walker provided me with much of the technical information required by this design. The network simulations were programmed and run by Leonard Zubkoff.

References

[1] Fahlman, S. E.
NETL: A System for Representing and Using Real-World Knowledge.
MIT Press, Cambridge, Mass., 1979.

[2] Fahlman, S. E.
The Hashnet Interconnection Scheme.
Technical Report, Carnegie-Mellon University, Department of Computer Science, 1980.

[3] Fahlman, S. E.
Preliminary Design for a Million-Element NETL Machine.
Technical Report, Carnegie-Mellon University, Department of Computer Science, 1980.
(forthcoming).

Table 1: Parts List

Device Type	Number
Custom 15 x 15 Selector (48 pin)	128
Custom 5x 1K Time Shifter (16 pin)	384
Custom 4x NETL Element Chip (48 pin)	240
64K Dynamic RAM (for selectors)	2176
64K Dynamic RAM (for shifters)	1920
2Kx8 Static RAM (for elements)	2160
Total Device Count	**7008**

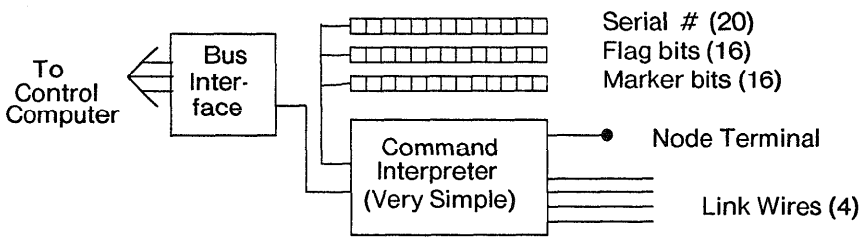

Figure 1: NETL Element

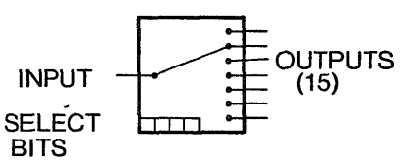

Figure 2A: The Selector Cell

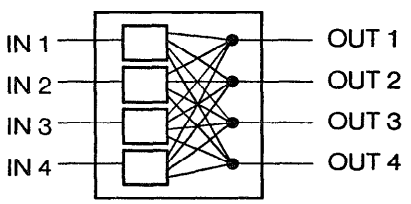

Figure 2B: Selectors on 15x15 Chip (Drawn as 4x4 chip.)

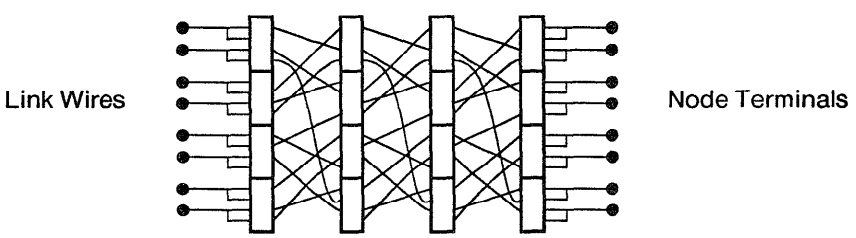

Figure 3: The Basic Hashnet Arrangement (simplified)