A Program For Mapping CNNs to Physical Architectures

Jim Bailey
Oregon Graduate Center
19600 N.W. von Neumann Dr.
Beaverton, Oregon 97006

Technical Report No. CS/E-88026
August 1988

Abstract

For simulation of systems with many processes on multi-processor architectures, it is necessary to partition the problem and assign subsets to each processor. In this paper, one such application, the emulation of Connectionist/Neural Networks (CNNs), is considered and a program to perform the required partitioning is described.

In addition to the description of the program design and usage, the theoretical background of the problem is presented along with a survey of some of the research on related problems. Finally, possible program enhancements and future research directions are described.

Table of Contents

CHAPTER 1

Introduction

In this chapter the problem of partitioning large Connectionist/Neural Network (CNN) systems and allocating the resulting pieces among multiple physical processors is introduced. This is followed by a brief introduction to the CNN computational model and a number of necessary definitions. Then two possible approaches to solving the mapping problem are presented and the chapter concludes with a brief analysis of the complexity of the mapping problem.

1. Introduction

The Cognitive Architecture Project (CAP) at the Oregon Graduate Center was created to develop and study architectures for efficient VLSI based emulation of very large Connectionist/Neural Network (CNN) systems. The mapping program presented here is designed to accept as input the description of a CNN and the target architecture it is to be mapped onto. The program is currently used in mapping CNN systems for simulation programs and ultimately will be used to assign the processors of emulation engines.

The mapping problem is an example of the class of graph embedding or partitioning problems and as such is NP complete. For the purposes of the CAP research effort, optimal solutions are not required so the heuristic approaches presented in this paper are feasible. These heuristics attempt to take into account what is known about both the source and destination graphs to improve the quality of the mapping while reducing the computational effort required.

2. The CNN Model

Although a knowledge of the CNN computational model is not required to understand the work described in this paper, the following brief presentation is provided as supplementary background material.

A CNN system consists of many simple processors or CNs (CNN nodes) communicating together to reach a consensus decision. Each CN independently and repeatedly calculates a new output and state function using its current state and inputs. The new output is in turn used by other CNs as input. In most models, the inputs are weighted before being used.

An example of a CN's update function is the $\sum - \prod$ computation illustrated in figure 1.1. Here each input is multiplied by the appropriate weight, the results are added together, and a sigmoid function is applied to the sum to generate the output. Currently, most CNN models use computations similar to the $\sum - \prod$

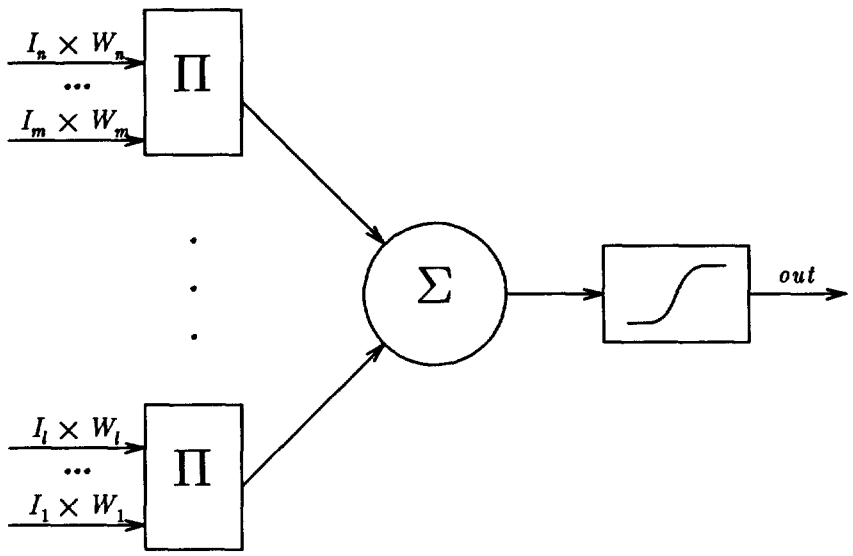


Figure 1.1 $\sum - \prod$ Computation

formula, but more complex in that historic values of inputs, various combinatorial groupings of the inputs, or the past state of the CN may be included in the computation.

In addition to the update or output function described above, most CNN models incorporate a learning function that adjusts the weights to change the behavior of the system over time. The major distinguishing characteristic of CNN models is that the information contained in a system is stored in the weights applied to the connections between CNs rather than being kept in named memory locations as the in typical algorithmic approach to computer architecture.

The CNN model is derived from the neuronal model of biology. Each neuron generates its output as a function of its past state and the inputs impinging upon it. Many researchers believe that memory and learning are the result of changes in how neurons affect each other. For a more complete introduction to the reasons for using the CNN model and some of the different approaches that have been used see [RuM86].

The precise computations used in a CNN model only impact the mapping process by setting lower limits on the required functionality of the physical processors and upper limits on the number of CNN nodes that can be effectively emulated by each physical processor. The major impact on the mapping process is from the

number and the physical dispersement of the CNs that are inputs to and outputs from each node. This interconnection pattern can be represented as a *directed graph* with CNN nodes as *vertices* and the dependencies between them as *edges* in the graph.

3. Definitions

This section presents some elementary definitions for use throughout the remainder of the paper.

Definition 1: A *directed graph* $G(V,E)$ is a set of vertices V and a set of edges E, where $E \subset V \text{x} V$. That is, each element of E is an ordered pair (i,j) where $i,j \in V$. The edge (i,j) is different from the edge (j,i) and is said to *originate* with vertex i and *terminate* with vertex j. □

For an example of a graph see figure 1.2 where the set of vertices is $\{a,b,c,d\}$ and the set of edges is $\{(a,b),(a,c),(a,d),(b,d),(c,a)\}$. The arrow heads on the edges indicate their direction. For example, (a,d) is an outgoing edge from vertex a and an incoming edge to vertex d. In this paper the term *node* is used as a synonym for vertex.

Definition 2: A *connection matrix* is an $N \text{x} N$ matrix where the i,j entry represents the edge from node i to node j in the related directed graph. □

Figure 1.3 shows the connection matrix for the graph of figure 1.2. Note that a 0 in a given position indicates the absence of a connection and a 1 indicates its presence. It is possible to use values other than 0 and 1, with the magnitude of the i,j entry indicating some attribute of the connection between i and j.

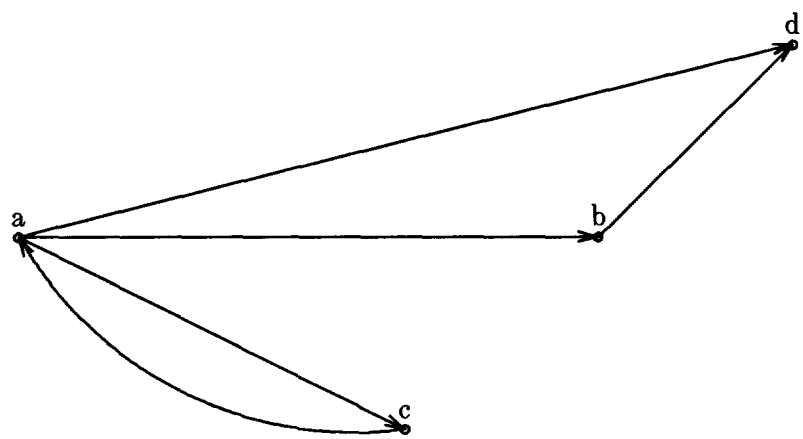


Figure 1.2 A Directed Graph

center;		ccccc.			a	b	c	d	a	0	1	1	1
b	0	0	0	1					c	1	0	0	0
d	0	0	0	0									

Figure 1.3 A Connection Matrix

Definition 3: A *c-graph* is the directed graph of a CNN where the CNs are represented by vertices and the connections between them are represented by corresponding edges. □

In a c-graph, the existence of the edge (p,q) implies that the output of CN p is an input to CN q. The edge (p,p) is not included in the c-graph, even if the past state of node p is an input to its update function, since the model is only of interprocessor communication. In general, c-graphs are asymmetrical, that is the existence of edge (p,q) does not imply the existence of edge (q,p). Asymmetry is not a requirement. Several CNN models, such as Hopfield networks [Hop82], have symmetrical graphs.

Definition 4: A *p-graph* is a directed graph representing the physical processor interconnect of a system. The vertices represent physical processing nodes and the edges represent the communication channels between them. □

Although the communication channel between any two PNs (physical nodes) p and q can be bi-directional, and is in most physical systems we are considering, in a p-graph this situation is represented by including both edges (p,q) and (q,p). This convention is used because the connection may consist of a pair of physically separate conductors or a single conductor together with switching circuitry to turn it around. Also, it allows for more general models than if only bi-directional connections were considered. The existence of local memory in, or connections between multiple CNs assigned to, node p is not represented by the edge (p,p), again because the model is of interprocessor communication only.

4. The Mapping Problem

While it is possible to simulate or emulate a CNN with a single processor system, enhanced performance requires the use of multi-processor systems. This need can be readily seen by considering the computations involved in updating a CNN consisting of 10^6 CNs with each CN connected to only 1% of the other CNs. With one of the simplest of the proposed computational models, the $\sum - \prod$ model presented earlier, there are a total of 10^{10} multiplications and 10^6 additions for a single network update, ignoring the time required to record the new state for each node. For a *real–time* system that needs to perform thousands of such network updates per second, a solution other than the use of a single processor must be found.

While the easiest and fastest method of storing the data for a uni-processor system is by having a single array of CN states that is referenced for each computation, this is not feasible for a multiple processor system. With every CN computation accessing the same structure, even with a sparse matrix, there will be a large amount of memory access contention. The solution is either to use a message passing multi-processor system or a variation on the Ultra architecture developed at NYU [GGK83]. A typical shared memory multi-processor system would not be appropriate, because of the high locality of memory references.

In order to effectively emulate a CNN on a message passing multi-processor system, it is necessary to partition the network and have each PN responsible for emulating a subset of the CNs. Two possible ways to partition a c-graph and to assign it to a p-graph are matrix splitting and graph embedding.

The first technique consists of considering the c-graph as a connection matrix and assign each PN a section of the matrix. Each PN would then calculate the partial sums for the CNs in its region and forward these values to designated PNs, such as those on the diagonal, for inclusion in the final computations. The computation completing PNs would then send new state values back and the cycle would repeat.

A disadvantage to this approach is that, with each CN's state distributed over multiple PNs, there is the potential for data incoherency. That is, there are multiple copies of each CN, with each possibly in a different state at the same time. To solve this problem requires synchronizing to insure that all copies of each CN are in the same state before updating calculations are performed. A second problem with the matrix partitioning approach is decreased fault tolerance. The loss of a PN deletes the computation of an entire section of the matrix and all rows and columns containing that section are disrupted.

Matrix partitioning is primarily of value in situations where PNs are powerful, so they are able to update multiple CN states easily; the system is synchronous, to reduce the probability of incoherent states; the connection matrix is uniformly dense, sparse connections reduce the efficiency of this approach and uniformity allows balanced use of all PNs; and interprocessor communication costs are expensive, so fewer, but longer, messages are preferable. Examples of message based multi-processor systems that would be appropriate for this approach are the BBN Butterfly and the Intel iPSC, with synchronizing messages broadcast system wide between node updates.

The second way to partition a c-graph is to assign one or more CNs to each PN and map each c-graph edge to the corresponding p-graph *path*. When a set of CNs has more *external* co-incident edges than the equivalent PN, either a group of PNs can be considered as a unit to provide the needed *fan-in/fan-out* or p-graph edges can be multiplexed and each c-graph edge mapped to a path in the p-graph. All intermediate PNs on a given path would only provide message forwarding, with no processing of messages not intended for them.

The primary thrust of CAP is to consider problems related to VLSI design, in particular the design of wafer-scale integrated CNN emulators, so processors are restricted in power and size, while the connections between them are fast and

inexpensive, although constrained in number. That is, the design specification requires maximal performance for the cost, ruling out the use of large complex processors connected with an intelligent interprocessor communication structure. For these reasons, together with fault tolerance considerations, the matrix partitioning approach will not be considered further.

5. Mapping Metrics

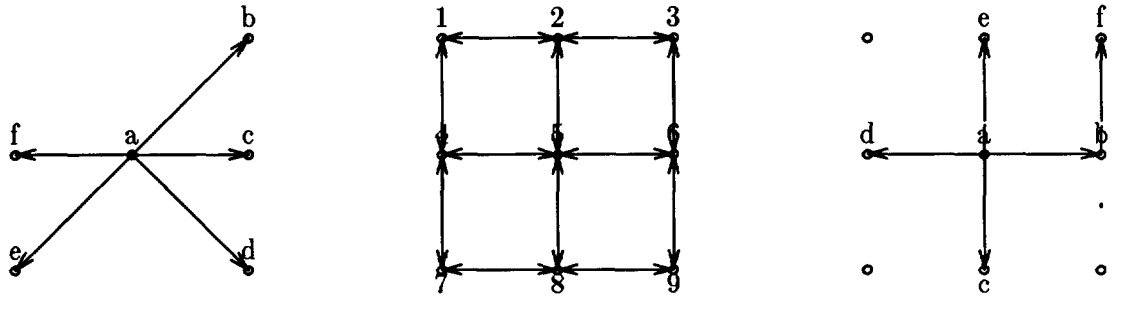
Since the goal is to find optimal mappings of c-graphs to p-graphs, it is necessary to define a measure of mapping *goodness* so that mappings may be compared. If each edge in the p-graph is assigned a *cost*, where the exact definition of cost is not specified, but might include such factors as power loss or transit time, then it is possible to sum the costs resulting from a particular mapping to provide a quantitative rating for that mapping.

Definition 5: A *path* in a graph is a sequence of edges $e_1,e_2,\cdots,e_n$ such that if $e_r=(i,j)$ and $e_{r+1}=(k,l)$ then $j=k$. The *beginning* and *end* of a path are the vertices v_1 and v_m where $e_1=(v_1,j)$ and $e_n=(k,v_m)$. As used in this paper, paths are restricted to include no loops. That is, no two edges of a path have the same starting or ending vertex. □

Definition 6: The *length*, $l(i,j)$, of a path is the number of edges it contains. □

Definition 7: A *mapping* $M:C\rightarrow P$, where C is a c-graph and P is a p-graph, is a function of $V_c\rightarrow V_p\bigcup Psi$ and $E_c\rightarrow P_p$ (P_p is the set of all paths in P) such that if $(i,j)\in E_c$ then either $M(i)=M(j)$ or $M(i,j)=p\in P_p$ and $M(i)$ is the beginning of p and $M(j)$ is its end. □

In other words, a mapping from a c-graph to a p-graph is an assignment of each CN to a PN. In addition, each edge connecting two CNs either disappears



c-graph c p-graph p $M:c\rightarrow p$

Figure 1.4 A mapping example.

(when both CNs go to the same PN), or is mapped to a path in the p-graph that connects the destination PNs and preserves the direction of the connection. An example of such a mapping is shown in figure 1.4. Vertex a has a fanout of five so that path {(5,6),(6,7)} is used to connect a and f, while all the other edges of c are mapped directly to edges of p.

Definition 8: The *cost* of a mapping M: $C \rightarrow P$ is $\sum_{e \in E_C} c(M(e))$ where e is an edge from E_C, the set of edges of C, and c is a function, that assigns a non-negative value to each path of P. □

If each edge traversed is assigned a cost of 1, then the mapping in figure 1.4 has a total cost of 6. This is the lowest possible mapping cost given the graphs c and p. A poorer mapping is $\{(a,1), (b,7), (c,8), (d,9), (e,6), (f,3)\}$. It would have a cost of $2 + 3 + 4 + 3 + 2$, or 14, twice the cost of the minimal mapping.

It is also possible to assign a cost to PNs based on size and complexity. Supporting more CNs with each PN will increase the area required, so the cost function of a mapping could include a PN cost function to show the effect of varying the CN/PN ratio. In the work presented here, it is assumed that there is an upper limit to the number of CNs per PN. All mappings that do not exceed this limit are given the same cost. This simplifies the calculations of cost and is sufficient for the purposes of this report.

The optimal, or target, mapping is the one with smallest such cost. To restate this in another way, the best mapping of a c-graph to a p-graph is the one in which each CN is placed to minimize the cost of the paths to all other CNs to which it is connected.

6. Mapping Complexity

In considering how to select an optimal mapping for a given c-graph and p-graph pair, it is instructive to realize the number of possible alternatives. Let C be the size of the c-graph, P be the p-graph size, and only map one CN to each PN. If $P > C$ there are a total of $C!$ possible mappings. Increasing the number of CNs per PN makes the number of possible mappings even greater.

The problem of determining an optimal mapping is extremely difficult with so many possibilities to consider. Clearly, the use of exhaustive search techniques is impossible for all but the smallest mappings. In chapter two several related problems and the heuristics that have been developed to provide usable solutions to them are presented.

7. Summary

The CNN model incorporates large numbers of communicating processes. For efficiency reasons, it is desirable to use multi-processor systems to emulate them. This requires some method of partitioning the CNN system. Two possible ways of doing this are graph embedding and matrix splitting, VLSI design considerations favor the use of graph embedding for the systems proposed by the CAP group.

In order to facilitate further discussion, the concept of a mapping was formally defined along with algorithms for assigning costs to mappings for comparative

ranking purposes. Finally, it was shown that the large number of potential mappings requires the use of heuristic approaches since no algorithm exists that provides a solution in polynomial time. Future chapters will present approaches that have been used for similar problems, suggestions for techniques that could be used here, and a description of how the mapper program is designed.

CHAPTER 2

Related Work

The mapping problem introduced in Chapter 1 is an example of the more general class of graph embedding or partitioning problems. In this chapter, other examples from this general set of problems are explored to see what approaches have been proposed for finding solutions. The solution techniques described here are all applicable to the mapping problem with slight modifications, but with varying degrees of efficiency.

1. Introduction

The CNN to physical processor assignment problem, when considered as a mapping from a c-graph to a p-graph, has similarities to many other graph partitioning and embedding problems. A variety of these problems have been studied with results published in the computer science literature. Examples of related problems include virtual memory utilization, partitioning of electronic circuits among a set of chips or boards, and assignment of processes to processors on multi-processor computers.

2. Iterative Improvement

The seminal treatment of the graph partitioning problem in computer science literature is the paper by Kernighan and Lin [KeL70]. Their problem statement is "given a graph G with costs on its edges, partition the nodes of G into subsets no larger than a given maximum size, so as to minimize the total cost of the edges cut." In the paper they show why a variety of solution techniques are invalid. The rejected approaches include generating a series of random solutions, use of max flow – min cut algorithms, identifying "natural clusters", and λ-opting.

The approach that they propose is to start with two subgraphs and recursively decrease the cost of the partitioning by swapping elements between them. The algorithm used to choose which elements to exchange is: calculate the potential change in cost for each pair of elements that could be swapped, pick the pair with the largest such change, set it aside on a list, and repeat the calculation and selection steps using the remaining elements until all pairs have been placed on the list in decreasing order of cost reduction. The final step is to swap all pairs that contribute the largest partial sum of differences. This sequence of steps is then repeated until no further gain is possible.

For a partitioning into more than two subgraphs, the graph is first divided into the required number of subgraphs, which are then processed by the above algorithm until they are pair-wise optimal. This approach is said to take order of n^2

time for each pass over all pairs of subgraphs, with another pass required to solve the problem of one pair of subgraphs needing adjustment after the exchange of elements in either one with a third. In their experiments, Kernighan and Lin found that for networks of size 100 or less divided into 6 or fewer subgraphs it took fewer than 5 passes to reach equilibrium.

3. Filling Modules

Russo et al [ROW71] examine the problem of partitioning a graph and assigning portions of it to modules. This is closer to the problem that this paper addresses. The difference is that the graphs that they have represent electrical or logic circuits and typically can be broken into graphs in a straight-forward manner. They show that the number of possible partitions is $\frac{n!}{p!^m m!}$, where there are n nodes in the graph, m modules to assign them to, and $m = n / p$.

Their approach is to generate a collection of subgraphs, then assign certain of these subgraphs to modules to eliminate the cost of the connections internal to each subgraph. All such assignments of subgraphs to modules are tried, with a recalculation of the mapping cost for each set of assignments. This approach is an improvement on the general problem because by limiting the choices of sets of nodes to predetermined subgraphs, rather than allowing arbitrary grouping, the number of possibilities to try is reasonable allowing exhaustive search. Unfortunately, their approach is not practical for the CNN mapping problem since the CN to PN ratio is so small compared with the total number of CNs in the system. The major contribution their work provided to the techniques of this paper was the statement that, in our terms, "use a CN to PN ratio that approaches the maximum possible to provide more efficient partitionings."

4. Use of Random Choices

Bokhari [Bok81] shows that the mapping problem as described here is equivalent to the graph isomorphism problem, the bandwidth reduction problem, and the quadratic assignment problem. He concludes that as the probability of ever finding an efficient exact algorithm is unlikely, research should concentrate on developing good heuristics. The heuristic presented in this paper is similar to that of Kernighan and Lin except that periodically random selections are used to pick the nodes to exchange. The use of occasional random choices helps the algorithm to avoid local minima. Bokhari's final conclusion though is "However, as the growth rate of time is bounded from below by N^2, the algorithm will probably not be suitable for very large arrays (say 32×32). For such arrays, entirely different heuristics will need to be developed."

5. Linear Time Heuristic

Fiduccia and Mattheyses [FiM82] have developed a variation on Kernighan and Lin that has worst case computation time per pass that grows linearly with the size of the problem. Their approach is similar to that of Russo *et. al.* in that it attempts to minimize the networks that are split between multiple blocks. Unfortunately, the CNN mapping problem has a fanout per CN much larger than the

number of CNs per PN so the technique is of limited value.

6. Mapping Parallel Algorithms

In their paper "On Mapping Parallel Algorithms into Parallel Architectures" [BeS84], Berman and Snyder show that the problem has two dimensions. The first is *cardinality variation* and in our terms means more CNs than PNs. The second dimension is *topological variation* and is the differences in the underlying graph structures.

Their approach is to first reduce the cardinality variation by *contracting* the algorithm graph. This is done by combining nodes with an adjacency preserving mapping. The problem is then reduced to a mapping from one graph to another of equal cardinality and is much easier. For the families of graphs they discuss in the paper the edge grammars with which they are defined provide a straight-forward automatable contraction algorithm. Some of the CNN models form similar families, such as the group of feed-forward layered networks with full interconnect between layers. For this family, a possible contraction algorithm would be to combine CNs that are in the same layer. This algorithm can be generalized for all feed-forward layered networks, by combining CNs that are in the same layer and share inputs or outputs. The mapping shown in figure 3.1 is an example of this algorithm carried to the ultimate where three *virtual nodes* replace the three layers.

This work has a great deal of promise as a possible technique for mapping well structured CNNs. Unfortunately, it requires a method of describing the structure of the CNN, such as edge grammars or layer definitions. For those situations where the structure of the CNN is either unknown or more arbitrary, the choice of which nodes to combine is equivalent to the original mapping problem in complexity. In any case, the concept of joining nodes based on their common connections to other nodes rather than on any connections between them is one that should be explored in future research on the mapper program.

7. Neural Network Mapping

Ghosh and Hwang, in their paper "Mapping Neural Networks onto Highly Parallel Multiprocessors" [HwG87], provide a good comparative analysis of a variety of computer architectures as tools for simulation of CNN models. Their mapping approach assumes that the c-graph has dense subregions with sparse interconnection between them. The algorithm is to cluster these regions onto adjacent PNs. Unfortunately, they do not describe a technique for recognizing these regions, nor do they mention how to handle graphs that are not of this nature.

8. Simulated Annealing

In his paper "Placement of Communicating Processes on Multiprocessor Networks" [Ste], Steele proposes the use of simulated annealing as a method of improving iterative mapping techniques. This approach is similar to that of Bokhari in that random choices are sometimes made during the selection process to avoid having the system settle into a local minimum. The difference is that with simulated annealing, the percentage of random choices starts high and decreases to zero as the network settles to a solution. It has been shown that simulated annealing will

generate solutions in general that are optimal or near optimal. The drawback is that the technique is even more computation intensive than iterative resolution alone and for this reason it is not feasible for our application due to the large sizes of the graphs involved.

9. Summary

The papers summarized in this chapter are only a small selection of the many that have been written on this subject. A number of authors have developed theoretical limits on the quality of mappings for different architectures. These papers were chosen to give a feel for some of the alternatives that have been proposed and to provide the reader with some pointers for further reading.

There are several basic approaches that have been proposed in the papers presented here. The first is to use an iterative algorithm, possibly with the addition of random choice to avoid local minima, another is to divide the network up into subgraphs that have some commonality and map these subgraphs, and the final is to compact the network by mapping it to another member of the same family of interconnect graphs. For the design of the system described in the remainder of this paper, the last two alternatives appear to have more applicability since iterative improvement algorithms scale non-linearly with the size of the problem. The cost of this choice is that the mappings generated are probably not as good as they would be otherwise. As explored in Chapter 4, if this cost becomes excessive, adding an iterative improvement algorithm as a second step after the initial assignment is a definite possibility.

CHAPTER 3

Mapper System Design

The mapper program accepts as input a *BIF* file and a *PAD* file. The output is either a list of node-to-node mappings or a *mapped BIF* file depending on the invoking arguments. In this chapter, I will first present the user interface, briefly describe the input and output file formats, and finally go into detail on the internal data structures and algorithms.

1. Usage

The mapper command line definition is:

mapper -p pad_file **-b** bif_file [**-dc**] [**-dp**]

The arguments can be in any order, with the first two required and the others optional. A **-p***ad* is followed by a space and the name or path of the input pad-file. Similarly, the **-b***if* is followed by a space and the name of the input bif-file. The **-d***ump* arguments are primarily intended for debugging purposes. They print the results of the mapping to *stdout*. That is, **-dc** generates a list of the CNs and which PN each is assigned to, while **-dp** prints a list of PNs and the CNs each contains. If there are no "-d" arguments, then the mapped bif is written to *stdout*. In any case, any error messages are written to *stderr*.

Thus, for most work, the proper usage is:

mapper -p pad_file -b bif_file > out_bif_file

or

mapper -p pad_file -b bif_file ¦ next_command

with the second form being preferred when *next_command* is capable of reading from *stdin*. The advantage of the second form is that disk space usage is reduced by not saving a copy of the temporary file *out_bif_file*. When other versions of mapper, that require more computation, are developed, the use of a temporary file may be more advantageous if the data it contains is to be used more than once and sufficient disk space is available for temporary file storage.

2. Input File Formats

Since both formats are defined formally in other papers, BIF in [Bah88] and PAD in [May88], this is an informal presentation and should not be taken as a definitive specification. For mapper, the information of interest in either a BIF or PAD file is the interconnection or graph description; most other data is ignored.

2.1. BIF

BIF (Beaverton Intermediate Format) is a CNN description language. It consists of a header section that lists the types of CNs present in the network

followed by a series of CN descriptions. Each CN description has a type indicator, an index that is a unique identifier in the network, a variety of state variables, and finally a collection of sites that indicate input and output dependencies. Each site is marked as being either input or output and broadcast or point-to-point. Each site is further broken down into links. A link consists of an index to identify it, a destination site, and other state information (ignored by mapper).

From this information, it is possible to construct a graph that shows the connections between CNs. The site value is kept to resolve multiple links to the same destination CN. Only output links are used in mapper's c-graph model. The c-graph is stored as a linked list of nodes. Each element in the list contains a pointer to a list of connections for that CN.

2.2. PAD

PAD (Physical Architecture Description) is a language that describes the physical system to be used to emulate the CNN. A PAD file begins with a header section that contains the number of PNs in the system, the maximum number of CNs per PN, and a variety of other descriptive values, not used in mapper. The header section is followed by an optional point-to-point (ptp) description. For each PN in the system the ptp section contains a list of the PNs it is directly connected to. The final two sections of the PAD file, also optional, are the PN region description and the physical broadcast hierarchy (pbh) section. PN regions are sets of PNs used as a convenience so that common sets do not have to be individually listed each time they occur. The pbh section lists groups of PNs (either individually or by region identifier) that have a common broadcast communication structure. A message sent by one PN in a group is received by all other members of the group.

Two separate lists are maintained by mapper to show the structure of the p-graph. One shows the nodes connected by ptp communication and the other shows those connected by broadcast. A more complete description is provided later in this chapter.

2.3. Mapping Hint File

The mapping hint file is still under development. It is intended to allow the designer of the BIF file to show how the c-graph is organized. This information may include a list of what subgraphs are appropriate for allocation to broadcast groups and is intended as an aid to efficient mapping of broadcast regions without requiring massive amounts of computation in mapper (also, the problem of deciding what CNs are "related" to others does not appear to be solvable in general).

3. Internal Data Structures

The mapper program is written in **C++** with **.h** files that define the classes of data structures and operators. Full program listings are provided in the appendix to this paper. There are three classes of objects: general purpose, c-graph, and p-graph. General purpose objects are definitions of elementary data structures that are used in defining the other two classes and in the utility sections of the program.

3.1. C-graph Objects

A c-graph is represented as a linked list of nodes. Each node contains the head of a linked list of connections. Each node is identified by a unique integer. A connection contains the identifier of the destination node along with the identifiers of the source and destination sites. This latter information is only used to check for duplicate connections and is not considered in determining connectivity.

In addition to the linked list, a hashed table of CNs is maintained so that a given node and its connection list can be located without scanning the CN list. This is strictly an efficiency device and does not impact the underlying structure.

The c-graph is accessed by a set of procedures:

get -- reads a BIF file and loads the c-graph structures

rank -- returns the number of CNs in the c-graph

build_tab -- builds the hash table

get_node -- returns a pointer to a CN

next_region -- returns a new subgraph (for disjoint subgraphs)

next_cn -- returns the closest cn to the previously gotten ones

log_pn -- logs a CN $\rightarrow$ PN assignment

dump_assigns -- prints all CNs with their assigned PNs

fil_bif -- reads the BIF file and writes it out, adding PN numbers and output types (ptp or broadcast)

3.2. P-graph Objects

The p-graph set of objects is similar to the c-graph set. There are two linked lists of nodes, one for those whose connectivity was defined by the ptp section of the file and the other for those defined by the pbh section. There is also a p-graph hash table to provide rapid access to the ptp list.

The p-graph accessing operators are:

get -- reads a PAD file and builds the linked lists

rank -- returns the number of PNs in the graph

cns2pn -- returns the maximum allowable CN to PN ratio

next_pn -- returns the next PN to assign CNs to

build_tab -- builds the hash table after get is done

get_node -- returns a pointer to a node given its identifier

4. Algorithms

The program consists of a main procedure that provides the logical flow to the system, a parser that reads a file and returns the next token, a set of utilities for list and table manipulation, and the procedures instantiating the c-graph and p-graph objects. The general flow of control is:

1) parse the command line
2) read the BIF file into the c-graph list
3) read the PAD file into the p-graph lists
4) build the c-graph and ptp access tables
5) perform the required mapping
6) dump the mapping results if requested
7) generate the mapped BIF otherwise

Any errors cause execution to terminate with an appropriate message.

The only section of the program that is detailed here is the mapping section. Command line parsing, file reading and data structure filling, data structure listing, and the insertion of mapping results in the BIF are all fairly straight-forward procedures executed in relatively obvious ways.

4.1. Broadcast Mapping

The broadcast regions of the p-graph are mapped first if they exist. This ordering of mapping reflects the feeling that broadcast provides a high potential degree of interconnect and should be used where feasible and not excessively expensive.

There are two considerations in mapping broadcast regions. The first is whether the p-graph region is large enough for all CNs that are to be assigned to it. The solution used in mapper is to maintain the region lists ordered by size and assigning the next largest c-graph region to the next largest p-graph region at each step. Although such a procedure may waste space by using p-graph regions that are larger than necessary, it is easy to implement and guarantees that as many c-graph regions will be mapped using broadcast as possible.

The second design consideration in mapping broadcast regions is how to effectively use overlapping or hierarchical regions when they are available. When the c-graph model consists of dense subgroups that are sparsely interconnected, the above selection process in adequate. For feed-forward layered networks, a structure common to many CNN models, allocating the layers to different broadcast regions to effectively use the overlapping feature is desirable. An example of such a mapping is shown in figure 3.1.

The information on p-graph structure, that is necessary in order to provide such a use of overlapping, can be presented in the PAD file by using the PN region convention. As described earlier, in a PAD file groups of PNs may be listed as belonging to the named regions and these regions used in turn to specify pbh groups. (This nomenclature is poor and should probably be changed with subregion

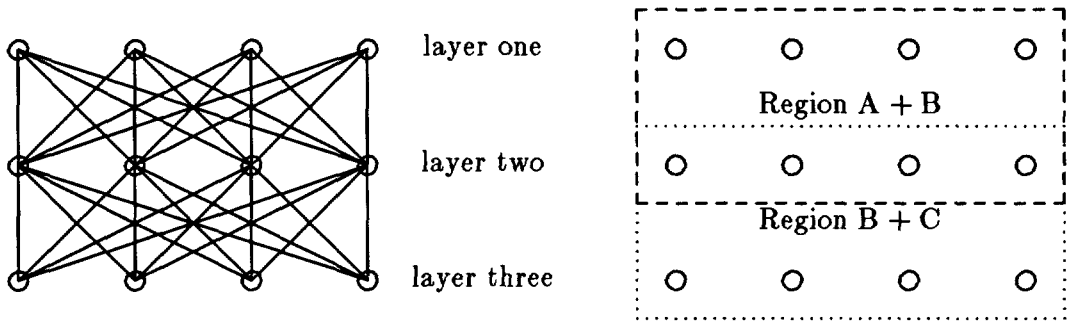


Figure 3.1 Mapping three layer feed-forward network to broadcast regions.

replacing region in future PAD definitions.) To map a feed-forward network, assign the CNs to regions such that pbh groups may be used to provide the communication between layers as well as any intra-layer connections. In figure 3.1, layer 1 is in region A, layer 2 is in region B, layer 3 in region C and the broadcast groups consist of each region alone for intra-layer communication, regions A & B combined for layer 1 to layer 2 connections, and regions B & C combined for layer 2 to layer 3 connections.

Other c-graph structures can be efficiently supported by judicious use of overlapping regions, but may require special case code in the mapper program together with structure specification in the hint file.

4.2. Point-to-point Mapping

The point-to-point communication mapping is done after any broadcast regions are filled. Its purpose is to provide for those edges in the c-graph that are not mapped in the broadcast or when there is no broadcast available. This portion of the mapper program is most similar to the other research presented in Chapter 2. To date, a simple, greedy heuristic has been sufficient for the networks and architectures being simulated. In the next chapter, possible alternatives are described.

The algorithm currently used assigns an average number of CNs to each PN. This approach was used since the goal was load leveling, to maximally use the Intel iPSC, rather than minimization of communication costs. If communication costs are to be reduced, each PN should be completely filled with CNs before going on to the next one. For load leveling, the number of CNs to assign to each PN is determined by dividing the total number of CNs by the total number of PNs. The choice of which goal to pursue, level load or minimal communication, is an option that may need to be changed based on the hardware characteristics. Later versions of mapper may implement the ability to vary this choice with either a command line

flag or an entry in the PAD file.

Mapper loops until all CNs are used. In each cycle, the *next* CN is assigned to the current PN. Whenever a PN reaches the predetermined limit, the *next* PN is selected and the process continues. New CNs are chosen from the set of CNs that are connected to those already mapped. When a set of CNs has been mapped, the number of edges to CNs not in the set are counted. If any outside CNs have multiple edges from the set, use the one with the most such connections. If two or more CNs have the same number of connections to the set, use the one with the most connections in common with the CNs of the set. Otherwise use the CN with the fewest additional connections outside of the set of already mapped CNs. When starting the algorithm, or when there are no CNs connected to the set, start with the CN and PN that have the largest fanout.

While the above algorithm provides good mappings for c-graphs that have a *spreading* pattern of connections, for example two dimensional grids, it fares badly with many layered networks; especially when there is a high degree of inter-layer connectivity coupled with a low degree of intra-layer connectivity. Networks of that form are better handled by the earlier described pbh approach. In the networks used for verifying mapper, it was usually 10% to 30% away from optimal mappings. These results were derived by examining ten small networks by hand and may not hold for larger, more complicated cases.

The major advantage of the current design is that the choice of which CN or PN to use next is isolated from the logic that does the mapping and keeps track of assignments. It would be very simple to plug in a different algorithm for either the PN or CN selection procedure.

5. Summary

The mapper program was designed to provide a flexible and easy to use interface for the user. In addition, the modular approach allowed by the use of C++ hides any knowledge of the input and output file structures in procedures that can be separately modified and maintained. Similarly, the choice of which subgraphs should be mapped together in broadcast mapping and which CNs go to which PNs is isolated from the code that provides the logical flow of the program. This use of objects and operators allows the implementor or maintainer to readily change one part of the system without having to worry about what impact the change will have on the remainder of the system.

CHAPTER 4

Future Directions

There are several open questions that could be answered by further research with the mapper program. They include comparing the efficiency of mapping various classes of c-graphs to different p-graph architectures, looking at the performance versus the cost of adding an iterative improvement step after the original assignment is completed, and which algorithms perform best for the initial mapping.

1. Mapping Comparisons

Before alternative approaches to mapping a particular c-graph and p-graph pair can be compared it is necessary to define the metrics to be used in the comparison. One simple way is to assign a cost of one to each p-graph edge and use the formula from the definitions section of Chapter 1 to calculate the cost of each mapping. Such a calculation could be implemented either as a final step in the mapper program or by a separate procedure that reads the mapped BIF and generates the result for each file. Installing the procedure as a subroutine in the mapper program would allow its use in implementing an iterative improvement technique.

One potential area of interest is how different mappings compare as the cost function is varied to reflect different values for broadcast versus point-to-point communications. Or another possibility is adding a weighting factor that allows for the increased size and complexity of PNs as the CN to PN ratio is increased. This second option would provide data on the effectiveness of calculation versus communication trade-offs.

2. Classes of Mappings

One critical research question is which p-graph architectures best support different classes of c-graph models. It is obvious that a p-graph can be tailored to efficiently support a particular c-graph. What is not readily apparent is what are the classes of c-graphs and what variations in p-graph structure best support them. What is the break-even point between the use of broadcast and point-to-point? Are there any p-graphs that effectively support a range of c-graph models?

These questions, along with similar ones, can be answered by mapping a wide variety of both c-graphs and p-graphs and comparing the costs. The CNN literature is a rich source of potential c-graph models to examine. One concern in the design of such research is verifying the quality of the mapper program so that the results reflect differences in the graphs rather than artifacts of the mapping process.

3. Iterative Improvement

Most of the papers reviewed in Chapter 2 describe some sort of iterative improvement technique for producing good mappings. The mapper program as currently implemented does not use such an approach. One problem is that adding an iterative step with the size of the c-graphs of interest, tens of thousands to millions of nodes, together with the number of separate groups that they are to be partitioned into, the number of PNs, may be prohibitively expensive. As indicated by Kernighan and Lin in their previously referenced paper, there are $(\frac{k}{2})$ pairs of subsets to consider, for k the number of PNs, and the time for a single pass through all such pairs is $O(n^2)$, where n is the number of CNs. They found that fewer than six passes were required for sets of 100 or fewer points divided into six or fewer sets. But, the number of passes is an increasing function of both the number of points and sets, it is possible that obtaining mappings close to optimal would require thousands of passes each taking O(10 sup 12) operations.

It would definitely be worthwhile to examine how much improvement a few passes would provide and how expensive it would be to do them. The most expensive part is calculating the cost of the exchange of the pairs of CNs and that can be reduced to counting the number of affected connections since the cost of swapping between two PNs is constant for each pair. Dividing the c-graph into subgraphs and repeating that procedure using an iterative approach at each point might have merit, but more work is needed on this question also.

4. Initial Selection Algorithms

The cost of performing a mapping is a function of both the number of CNs and PNs. For this reason a division of the initial c-graph into a set of subgraphs that are then separately processed would favorably impact the mapping performance. What the impact would be on quality is an open question as well as what criteria would be used to make the divisions. Given some information on c-graph structure in the "hint" file would provide a starting point for such a divide and conquer algorithm.

The area with the best potential for improvement is replacing the current point-to-point mapping algorithm. Would it be better to keep adding the CN that has the most connections at every step? What about the use of input connections as well as the output connections currently being used? As is mentioned in the discussion on the Berman and Snyder paper, there might be more value in choosing CNs that are not connected to the ones already chosen, as is done currently, but choose ones that have connections to common sources or destinations. Should some random selection be made when two or more CNs are potential choices to overcome the bias from the original BIF ordering? Another possibility is use the connectivity matrix and swap rows and columns until non-zero entries are grouped together. These dense regions would then be mapped to neighboring PNs.

5. Conclusions

Although researchers have done much work in developing techniques for solving graph partitioning and mapping problems, the particular area of CNN mappings

remains largely unexplored. Possible areas for future research include determining classes of c-graphs and the p-graph structures best suited for each class and methods of improving the quality of the mappings.

CHAPTER 5

Summary and Conclusions

In this paper some of the underlying theory for the mapper design has been presented together with the implementation detail of the programs. Related research has been reviewed and possible future research directions detailed.

1. Summary

CNN models require larger and faster implementations to provide adequate performance for many real world problems. The use of parallel computational systems is a means to such improved performance. Parallel systems require partitioning of the problem. Two partitioning techniques are graph mapping and matrix splitting. Graph mapping is better suited to VLSI implementation. The graph mapping problem has too many possible solutions to consider all and pick the best, some sort of heuristics must be used to generate adequate solutions. Two heuristics have been developed, one for mapping of pre-specified groups of CNs to broadcast regions and the other for individual mapping of CNs to PNs for point-to-point communications. They have proven to be adequate for the networks examined so far, but several possible future directions for improvement are suggested.

In designing the mapper program the use of the C++ language together with the concept of objects and operators on them has been used to isolate implementation details, file structures, node and region selection algorithms, and the logical organization of the program. This design allows for easy modification of some subpart of the program without requiring rewriting of other sections. It also facilitates the plugging in of new algorithms for testing purposes.

2. Conclusions

The mapper program as it currently exists provides a solution to the need to partition CNN networks among systems of multiple processors. It is designed to be readily upgraded when research indicates the appropriate directions for improvement. The performance has been adequate, without requiring large amounts of computation time or space. This program is not presented as being optimal, just as being a working solution to the problem.

Bibliography

[Bah88] Bahr, C., "ANNE, A General-Purpose Neural Network Emulator for the Intel iPSC Hypercube," CSE Technical Report, Dept. of Computer Science/Engineering, Oregon Graduate Center, Beaverton, OR, March 1988. In preparation..

[BeS84] Berman, F. and Snyder, L., "On Mapping Parallel Algorithms into Parallel Architectures," *Proc. Int. Conf. on Parallel Processing*, 1984, pp. 307-309.

[Bok81] Bokhari, S. H., "On the Mapping Problem," *IEEE Transactions on Computers*, vol. c-30(March 1981), pp. 207-214.

[FiM82] Fiduccia, C. M. and Mattheyses, R. M., "A Linear-Time Heuristic for Improving Network Partitions," *Proc. 19th Design Automation Conference*, 1982, pp. 175-181.

[GGK83] Gottlieb, A., Grishman, R., Kruskal, C. P., McAuliffe, K. P., Rudolph, L. and Snir, M., "The NYU Ultracomputer - Designing a MIMD Shared Memory Parallel Computer," *IEEE Transactions on Computers*, vol. C-32(February 1983), pp. 175-189.

[Hop82] Hopfield, J. J., "Neural networks and physical systems with emergant collective computational abilities," *Proc. Nat. Acad. Sci. USA*, vol. 79(April 1982), pp. 2554-2558.

[HwG87] Hwang, K. and Ghosh, J., *Mapping Neural Networks onto Highly Parallel Multiprocessors*, University of Southern California, 10/1/87.

[KeL70] Kernighan, B. W. and Lin, S., "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell System Technical Journal*, February 1970, pp. 291-307.

[May88] May, N., "Fault Simulation of a Wafer-Scale Neural Network," Tech. Report CS/E-88-020, Dept. of Computer Science/Engineering, Oregon Graduate Center, Beaverton, Oregon, May 1988.

[RuM86] D. E. Rumelhart and J. L. McClelland, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1 and 2, Bradford Books/MIT Press, Cambridge, MA, 1986.

[ROW71] Russo, R. L., Oden, P. H. and Wolff, P. K., "A Heuristic Procedure for the Partitioning and Mapping of Computer Logic Graphs," *IEEE Transactions on Computers*, vol. c-20(December 1971), pp. 1455-1462.

[Ste] Steele, C. S., "Placement of Communicating Processes on Multiprocessor Networks," Technical Report 5184:Tech. Rep.:85, California Institute of Technology.