Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

Implementing A Tool For Designing Portable Parallel Programs

by
Geetha Chitti

The Implementation aspects of a novel parallel programming model called Cluster-M is presented in this thesis. This model provides an environment for efficiently designing highly parallel portable software. The two main components of this model are Cluster-M Specifications and Cluster-M Representations. A Cluster-M Specification consists of a number of clustering levels emphasizing computation and communication requirements of a parallel solution to a given problem. A Cluster-M Representation on the other hand, represents a multi-layered partitioning of a system graph corresponding to the topology of the target architecture. A set of basic constructs essential for writing Cluster-M Specifications using PCN are presented. Also, a C program for generating the Cluster-M Representations is shown. Cluster-M Specifications are to be mapped onto the Representations using a proposed mapping methodology. Using Cluster-M a single software can be ported among various parallel computing systems. This thesis concentrates on the implementation of the Specifications and the Representations.

IMPLEMENTING A TOOL FOR DESIGNING PORTABLE PARALLEL PROGRAMS

by
Geetha Chitti

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer and Information Science

May 1993

APPROVAL PAGE

Implementing A Tool For Designing Portable Parallel Programs

Geetha Chitti

Dr. Mary M. Eshaghian, Thesis Advisor (date)
Assistant Professor of Computer and Information Science, NJIT

Dr. Daniel Y. Chao, Committee Member (date)
Assistant Professor of Computer and Information Science, NJIT

Dr. David Wang, Committee Member (date)
Assistant Professor of Computer and Information Science, NJIT

BIOGRAPHICAL SKETCH

Author: Geetha Chitti

Degree: Master of Science in Computer Science

Date: May 1993

Undergraduate and Graduate Education:

- Master of Science in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 1993

- Bachelor of Science in Electronics and Communication Engineering,
  Sri Venkateswara University, Tirupathi, India, 1988

Major: Computer Science

This thesis is dedicated to my family

ACKNOWLEDGMENT

The author wishes to express her sincere gratitude to her supervisor, Dr. Mary M. Eshaghian for her guidance, friendship, and moral support throughout this research.

Special thanks to Dr. Daniel Chao and Dr. David Wang for serving as members of the committee. The author is grateful to the Department of Computer and Information Science for partial funding of the research.

The author appreciates the timely help and suggestions from the project group team members Phil Chen, Ying-Chieh Jay Wu and Ajitha Gadangi.

The author appreciates Sekhar Chitti for providing his precious time and expert assistance in producing this document.

Last but not the least the author would like to thank her beloved husband Sarma Chitti for his love, support and encouragement.

TABLE OF CONTENTS

Chapter Page

LIST OF FIGURES

CHAPTER 1
INTRODUCTION AND BACKGROUND

The task of designing parallel algorithms for specific architectures is difficult. Every algorithm is specific to the particular architecture. In this thesis, we focus on implementing a tool that enhances this process of mapping the Specification(algorithm) to the Representation(architecture). In the following we give a brief introduction to parallel architectures, algorithms and issues related to efficient mapping techniques. In the rest of the thesis, we give an introduction to the Cluster-M components first, and then discuss the implementation aspects for each of these components.

1.1 Parallel Architectures

The characteristics of parallel algorithms are intimately interwined with the characteristics of the problem to be solved and the computer architecture on which the algorithm will be implemented. We use the term "architecture" to include the programming environment and operating system support, as well as machine hardware. However, the most significant characteristic of parallel architectures is the organization of memory, specifically whether each processor has access only to its own private local memory, or memory is globally shared among all processors. A basic uniprocessor architecture

has three major components: the main memory, the central processing unit(CPU), and the input/output subsystem. A multiprocessor architecture consists of two or more uniprocessors. These architectures are classified into three schemes: Flynn's classification, which is most widely used is based on the multiplicity of instruction streams and data streams. These are SISD(Single Instruction stream -Single Data stream), SIMD(Single Instruction stream -Multiple Data stream), MISD(Multiple Instruction stream -Single Data stream), MIMD(Multiple Instruction stream -Multiple Data stream). Feng's classification is based on serial versus parallel processing, Handler's classification is determined by the degree of parallelism and pipelining. The SIMD systems are currently being used for scientific operations as their are especially suitable for exploiting the parallelism inherent in certain tasks. In designing SIMD systems, constructing an interconnection network for communications among the processors and memories presents a major problem.

In designing the architecture of an interconnection network four design decisions can be identified. They concern operation mode, control strategy, switching method, and network topology. The operation modes are classified into three categories: Synchronous, Asynchronous and Combined. All existing SIMD machines choose the synchronous operation mode. The control strategies are classified as centralized control and distributed control. Most existing SIMD networks choose the centralized control on all switch elements by the control unit. The two major switching methodologies are cir-

cuit switching and packet switching. Last of all, based on network topologies the SIMD interconnection networks are classified into two categories: Static networks and Dynamic networks. In a static network, links between two processors are passive and dedicated buses cannot be reconfigured for direct connections to other processors. Examples of static network topologies are linear array, ring, star, tree, near-neighbor mesh, systolic array, completely connected, n-cube and cube-connected cycle as shown in Figures 1, 2. In a dynamic network, links between two processors can be reconfigured by setting the network's active switching elements. Examples of dynamic network topologies are single stage, multistage, and crossbar as shown in Figure 3.

1.2 Parallel Algorithms

An algorithm performs a single well defined function. A task is performed by execution of a collection of algorithms. The task of designing parallel algorithms presents challenges that are considerably more difficult than those encountered in the sequential domain. The lack of a well-defined methodology is compensated by a collection of techniques and paradigms that have been found effective in handling a wide range of problems. This section introduces these techniques which are interesting on their own and often appear as subproblems in numerous computations. The techniques are balanced binary tree, the pointer jumping technique, divide-and-conquer technique and the pipelining technique.

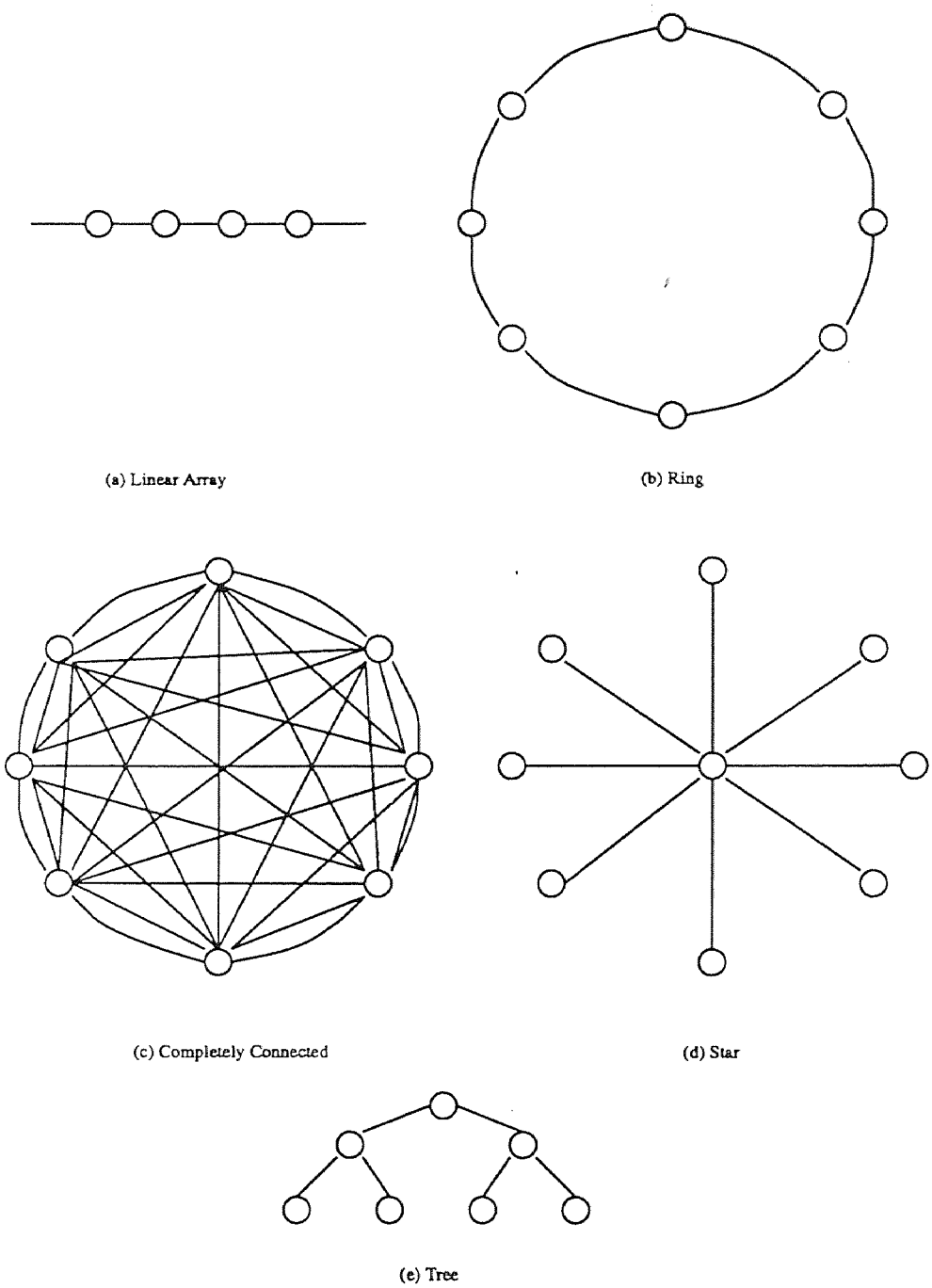
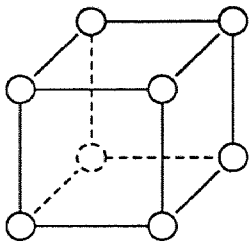
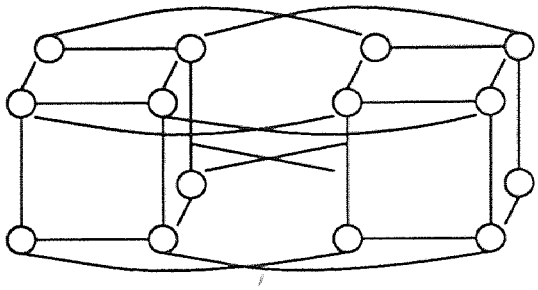


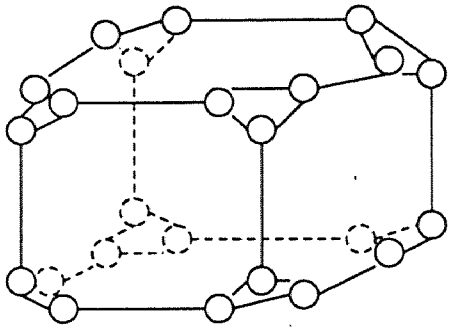
Figure 1. Static Interconnection Topologies



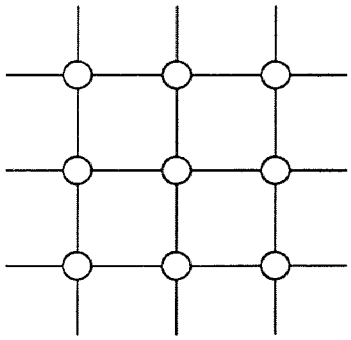
(f) 3-cube



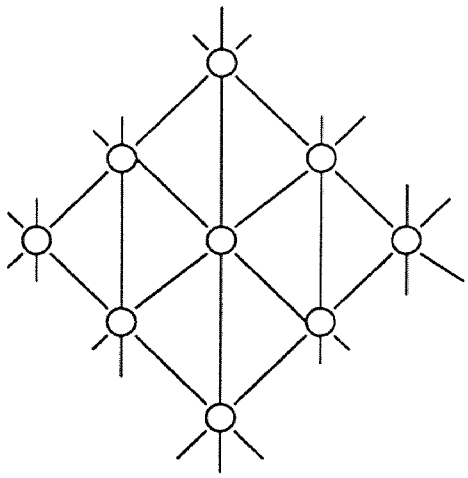
(g) 4-cube



(h) 3-cube-connected cycle



(i) Nearest-neighbor mesh



(j) Systolic array

Figure 2. Static Interconnection Topologies

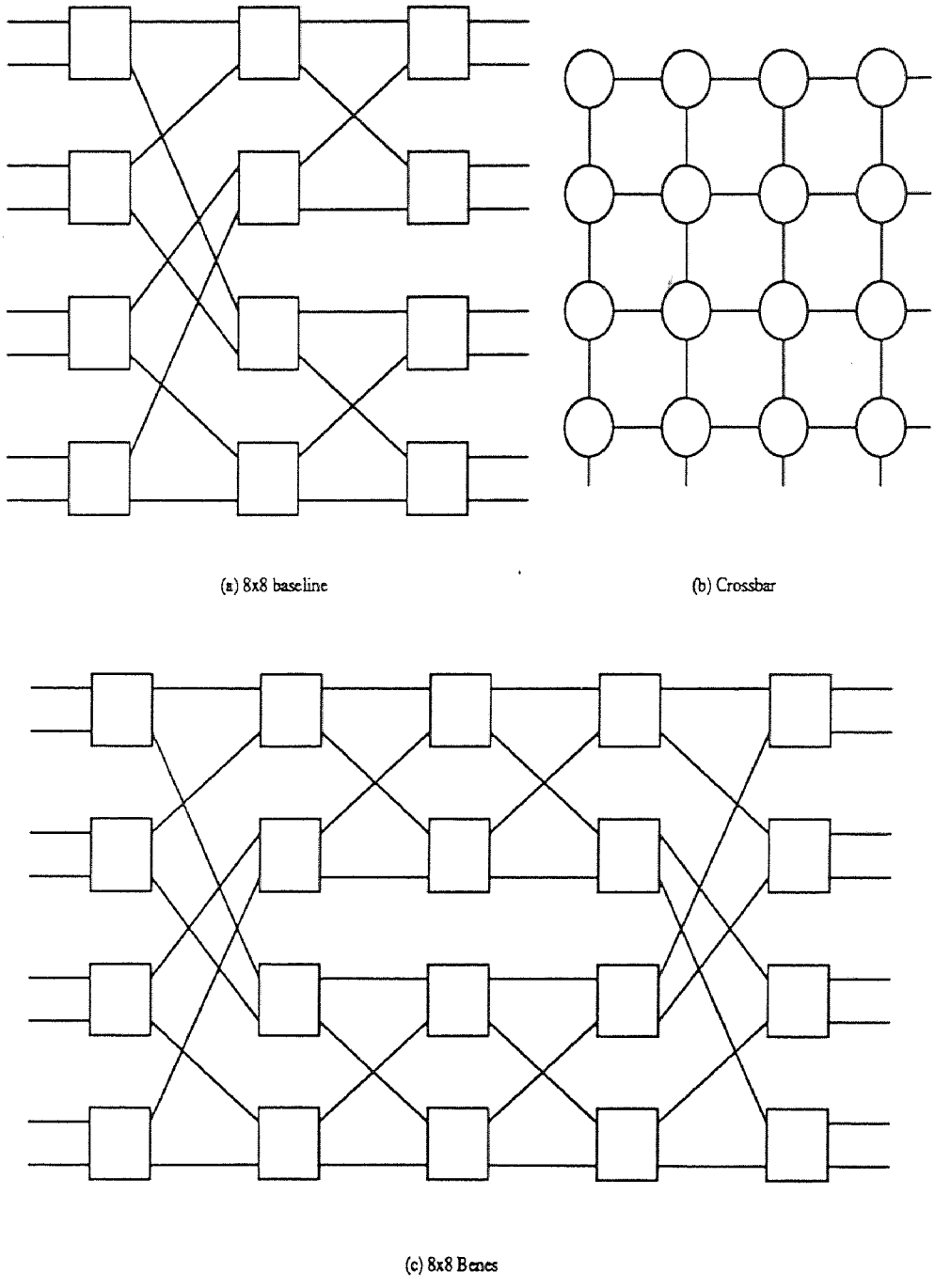


Figure 3. Dynamic Interconnection Topologies

The basic scheme to build a balanced binary tree on the inputs and to traverse the binary tree to or from the root leads to efficient algorithms for many simple problems. This scheme is one of the most elementary and the most useful parallel techniques. Broadcasting a value to all the processors, and compacting the labeled elements of an array, are two simple examples that can be handled efficiently by this scheme. The pointer jumping technique provides a simple and powerful method for processing data stored in linked lists or directed rooted trees. The pointer jumping technique is useful in general because it is simple and can effectively handle subproblems arising in many computational tasks. These subproblems are usually of a size small enough that the pointer jumping technique will allow optimal overall processing. It is also possible to use the pointer jumping technique in combination with other techniques to achieve optimality. The divide-and-conquer strategy constitutes a powerful, widely applicable approach for developing efficient parallel algorithms. However, a straight-forward divide-and-conquer approach does not lead to optimal O(log n) time algorithms, unless the merging can be performed effiently. Pipelining is an important parallel technique that has been used extensively in parallel processing. In the next section, a brief introduction on portable software is presented.

1.3 Portable Software

A Highly parallel software is usually designed to be suitable for executing on specific target multiprocessor system. Adapting such pack-

ages to run on different machines may require a complete re-write, a time-consuming endeavor. Therefore, it is desirable that a software package be portable and executable among various architectures. Certain tools are needed to act as intermediate media based on which machine-independent algorithms can be designed. Such programming tools will also provide mechanisms for mapping a given program onto desired underlying architectures.

One of the parallel programming models extensively described in literature is *Linda* [1][1] . Linda defines a logically shared data structuring memory mechanism called tuple space. Tuple space holds two kinds of tuples; process tuples which are under active evaluation, and data tuples that are passive. Ordinarily, building a Linda program involves dropping a process tuple into tuple space spawning off other process tuples. This pool of process tuples, all executing simultaneously, exchange data by generating, reading, and consuming data tuples. A process tuple that has finished executing turns into a data tuple, indistinguishable from other data tuples. Once a program is written based on the Linda model, each step must get implemented using the underlying architecture. Linda requires large volumes of data exchanged to and from the shared memory which may lead to heavy congestion over available communication channels of a typical multiprocessor system. For this reason, Linda has been mostly used for coarse grain computations. Furthermore, it is very difficult to implement Linda on architectures not supporting the shared memory structure.

[1]Parenthetical references placed superior to the line of text refer to the bibliography.

In contrast to Linda, the programming model *Express* supports a distributed memory system organization. The Express paradigm provides a parallel programming language which allows the user to specify the names of processors supposed to exchange information. Express handles the routing without requiring the user to specify the routing path or algorithm. Express also contains some built in constructs which can translate certain forms of a sequential program into its parallel equivalent. However, the algorithms coded using Express are machine dependent and therefore are not fully portable.

A few other examples of parallel programming models are: the *Actors* Programming model [2], and *Tool for Large-Grained Concurrency (TLC)*. TLC, developed by BBN, employs a language based on common-LISP with implicitly parallel constructs to specify the dependencies among a set of coarse-grained remote computations. The TLC compiler translates a TLC program into a network of "continuations", separated by object-oriented invocations on remote servers which encapsulate the bulk of the simulation processing behind abstract interfaces. The TLC virtual machine, which typically runs on the end-user's workstation, executes the program by sequentially selecting and executing an eligible continuation from the run queue until it is empty. Unfortunately, this sequential bottleneck prevents the algorithm from being executed efficiently in parallel. The model Actors, on the other hand, allows massive parallel execution of algorithms since it consists of self-contained, interactive, and independent components of a computing system that communicate by asynchronous

message passing. At an overhead cost of implementing such system, Actors is machine independent: it can be executed on shared memory computers as also over distributed networks.

In this thesis, we study the implementation aspects of a novel parallel programming model called *Cluster-M* which allows parallel programs to be written independent of underlying structure. Cluster-M has two main components: the Cluster-M, Representations, and Cluster-M Specifications of a problem. The Cluster-M Representation of an architecture incorporates the processor interconnection topology. A parallel program executable by this model is called the Cluster-M Specification which represents the communication and computation needs of a solution to the problem. The Cluster-M Specification will then be mapped onto the Cluster-M Representation of the underlying architecture using Cluster-M mapping module. The same Specification may be used for any other form of Cluster-M Representation. Cluster-M provides efficient means for designing portable algorithms which can be mapped onto various multiprocessor organizations.

The rest of the thesis is organized as follows to describe the different components of Cluster-M in detail. In Chapter 2, we present the Components of Cluster-M.

In Chapter 3, we present PCN implementation of seven Cluster-M constructs and macros essential for writing portable Cluster-M Specifications. Also an efficient algorithm for generating the PCN Representations is presented. In Chapter 4, the Cluster-M mapping

module is studied and one of the implementation aspects of it is proposed and an application of Cluster-M to heterogeneous computing is discussed and an example is presented. In Chapter 5, a conclusion and future research is presented.

CHAPTER 2
COMPONENTS OF CLUSTER-M

In this section we present the components of Cluster-M which are the Cluster-M Specifications and the Cluster-M Representations.

2.1 Cluster-M Specifications

A Cluster-M Specification of a problem is a high level machine-independent program that specifies the computation and communication requirements of a given problem. A Cluster-M Specification consists of multiple levels of clustering. In each level, there are a number of clusters representing concurrent computations. Clusters are merged when there is a need for communication among concurrent tasks. For example, if all n elements of an array are to be squared, each element in a cluster, then the Cluster-M specification would state:

For all n clusters, square the contents.

Note, that since no communication is necessary, there is only one level in the Cluster-M Specification. The mapping of this Specification to any architecture having n processors would be identical. Using the Cluster-M constructs presented in the next section, the above example can be written as follows:

The Cluster-M specification of a given problem consists of several layers of clusters with the lowest layer consisting of clusters each

containing a single computation operand. This similarity between Cluster-M representation and specification results in simplification of mapping problems to architectures and the means to design portable algorithms. This will be more evident in the next section where mapping strategies are discussed.

All initial Cluster-M clusters involved in a computation are merged into one cluster in the next clustering level. Clusters in intermediate levels are merged, split, and/or their elements manipulated according to computation and communication requirements. These operations on the clusters of each level, unlike dataflow paradigm, are level independent.

The basic operations on the clusters and their contained elements are performed by a set of constructs which form an integral part of the Cluster-M model.

The following is a list and description of the constructs essential for writing Cluster-M Specifications.

- $CMAKE(LVL, x, ELEMENTS)$

  This construct creates a cluster x at level LVL which contains ELEMENTS as its initial elements. ELEMENTS is an ordered tuple of the form $ELEMENTS = [e_1, e_2, \cdots, e_n]$ where n is the total number of components of ELEMENTS. The components of ELEMENTS could be scalar, vector, mixed-type, or any type of data structure required by the problem.

- $CELEMENT(LVL, x, j)$

  This construct yields the j-th element of cluster x of level LVL. If j is replaced by '-', then CELEMENT yields all the elements of cluster x. If x is replaced by '-', then CELEMENT yields all the elements of all clusters of level LVL.

- $CSIZE(LVL, x)$

  Yields the number of elements of cluster x, (i.e. $\|CELEMENT(LVL, x, -)\|$).

- $CMERGE(LVL, x, y, ELEMENTS)$

  This construct merges clusters x, y of level LVL into cluster $\min x, y$ of level $LVL + 1$. The elements of the new cluster are given by ELEMENTS. If ELEMENTS in CMERGE is replaced by '-', the elements of the new cluster are given by $[CELEMENT(LVL, x, -), CELEMENT(LVL, y, -)]$ (i.e. the elements of x are concatenated to the elements of y to form ELEMENTS of the combined cluster).

- $CUN(LVL, *, x, i)$

  This construct applies unary operation $*$ to the i-th element of cluster x. If i is replaced by '-', then the operation is applied to all elements of x. If both i and x are set to '-', then the operation is applied to all elements of all clusters of level LVL.

- $CBI(LVL, *, x, i, y, j)$

  This construct applies binary operation $*$ to the i-th element of cluster x and the j-th element of cluster y. If i, j are replaced

by '-', then the binary operation is applied to all elements of x, y. CBI returns the resulting components.

- $CSPLIT(LVL, x, k)$
  This construct splits cluster x of level LVL at k-th element into two clusters of level LVL+1.

Using these constructs the previous problem specification can be written as:

begin

LVL=1
$CUN(LVL, Square, -, -)$

end

2.2 Cluster-M Representations

For every architecture, at least one corresponding Cluster-M Representation can be constructed. Cluster-M Representation of an architecture is a multi-level nested clustering of processors. To construct a Cluster-M Representation, initially, every processor forms a cluster, then clusters which are completely connected are merged to form a new cluster. This is continued until no more merging is possible. In other words, at level LVL of clustering, there are multiple clusters such that each cluster contains a collection of clusters from level

$LVL-1$ which form a clique. At the highest level there is going to be only one cluster, if there exists a connecting sequence of communication channels between any two processors of the system. A Cluster-M Representation is said to be *complete* if it contains all the communication channels and all the processors of the underlying architecture. For example, the Cluster-M Representation of the n-cube architecture is as follows: At the lowest level, every processor belongs to a cluster which contains just it self. At the second level, every two processors (clusters) which are connected are merged into the same cluster. At the third level, clusters of previous level which are connected belong to the same cluster, and so on until level n. The complete Cluster-M Specification of a 3-cube, a 2×4-mesh, a ring of size 8, completely connected system of size 8, and a system with arbitrary interconnections are shown in Figures 4, 5, 6, 7, 8 respectively.

A Cluster-M Representation with k nested subcluster levels represents a connected network of processors with diameter $\Omega(k)$. To investigate the relationship between the clustering levels of an architecture and its diameter, lets define D_{LVL} the diameter of Cluster-M Representation at clustering level LVL. D_{LVL} is defined as the maximum number of communication steps needed between any two processors contained in any single cluster at level LVL.

The diameter of the Representation at level $i+1$ can be expressed as:

$$D_{LVL+1} = D_{LVL} + (\text{communication overhead of level } LVL+1)$$

For example, let us consider a ring-connected architecture with N processors where $k = \log N$ levels. The Cluster-M Representation for this architecture is given in Figure 6. In this case every two adjacent clusters will be merged, the size of clusters is doubled at level LVL compared to $LVL - 1$. $k + 1$ such levels result. The diameter of the network can be found by examining D_{LVL} for several levels:

$$D_2 = 2 - 1$$
$$D_3 = 4 - 1$$
$$D_i = 2^{i-1} - 1$$

Thus at the maximum level $k = \log N$, the network diameter $= O(2^k)$. The relationship between network diameter and the number of clustering levels depend on the degree of connectivity of the processor nodes and on connection patterns at each level.

Before presenting an algorithm to find Cluster-M Representations, we define several terms and identify some clustering properties:

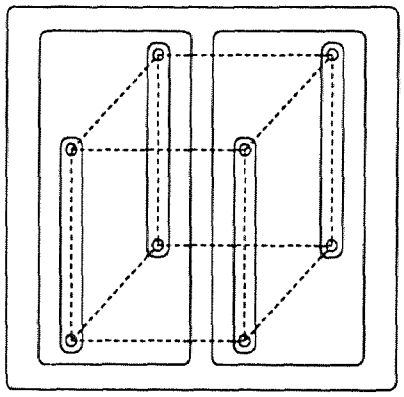


Figure 4. Cluster-M Representation of N-Cube of Size 8.

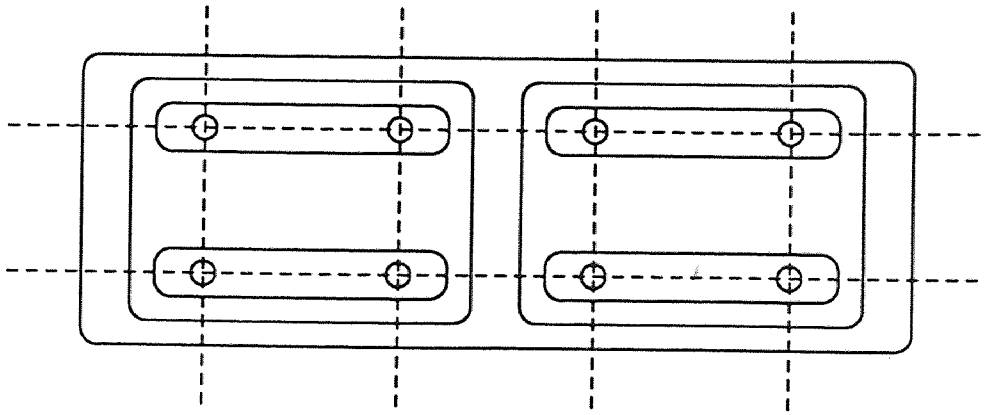


Figure 5. Cluster-M Representation of Mesh of Size 8.

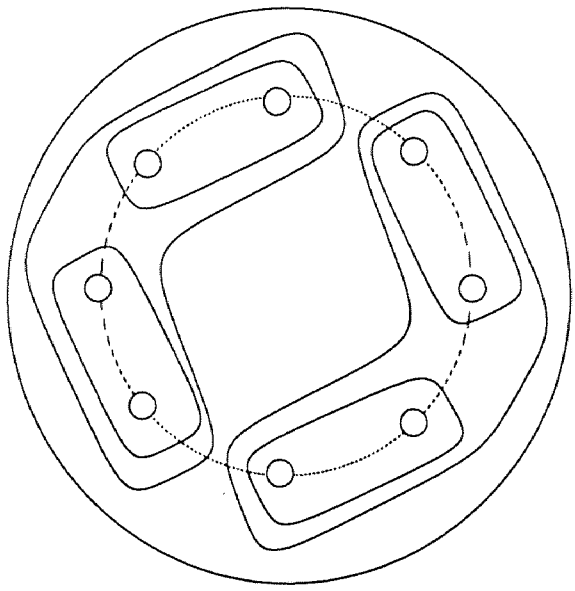


Figure 6. Cluster-M Representation of a Ring of Size 8.

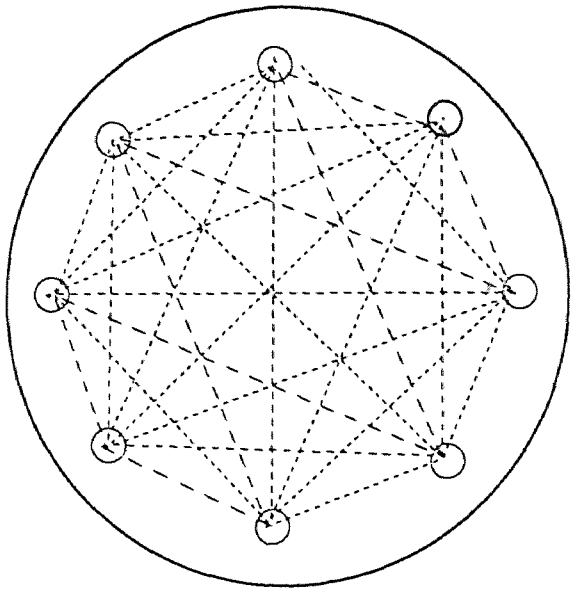


Figure 7. Cluster-M Representation of A Completely Connected System of Size 8.

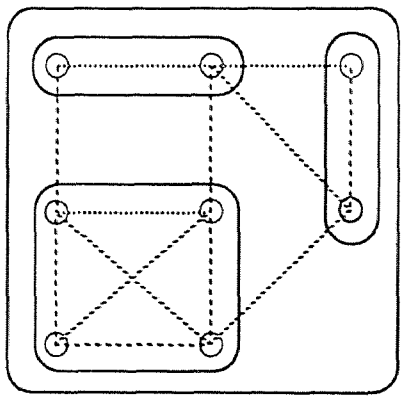


Figure 8. Cluster-M Representation of An Arbitrarily Connected System of Size 8.

- The system graph of an N-processor system $S = (P, E)$ is an undirected graph represented by the adjacency matrix, where $A(i, j) = 1$ indicate a communication link between processors i, j, $1 \leq i, j \leq N$.

- A clique in an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices each pair of which is connected by an edge in E. In other words, a clique is a complete subgraph of G.

- A system processor is contained in only one cluster at level LVL. Let $PC(LVL, x)$ designate all processors belonging to cluster x of level LVL. Thus for clusters x, y of level LVL, $PC(LVL, x) \cap PC(LVL, y) = \phi$. At $LVL = 1, PC(1, x) = x$.

- Each cluster is identified by the lowest numbered processor contained in the cluster (i.e for cluster x, $x = \min PC(LVL, x)$). Thus let $CLUSTERS(LVL) = [c_1, \cdots, c_m]$ be an ordered tuple designating the clusters at level LVL, with m being the number of such clusters.

- The clusters of level LVL form an undirected graph where two clusters x, y are connected if there exists processors $p_x \in PC(LVL, x)$, and $p_y \in PC(LVL, y)$, where $A(p_x, p_y) = 1$.

- Define $C(LVL, p) = c$ to indicate that processor p belongs to cluster c of level LVL, $1 \leq LVL \leq k$, where k is the maximum number of clustering levels .

With the aid of the above properties and definitions, we next present an algorithm to generate Cluster-M system Representation.

CHAPTER 3
IMPLEMENTATION OF COMPONENTS

In this section, we first give a brief introduction to Program Composition Notation (PCN), a parallel programming system selected as an implementation medium for the various components of Cluster-M. We then discuss Cluster-M components implemented in PCN.

3.1 Program Composition Notation (PCN)

Program Composition Notation is a system for developing and executing parallel programs [4]. It comprises of a high-level programming language, tools for developing and debugging programs in this language,and interfaces to Fortran and C that allow the reuse of existing code in multilingual parallel programs. Programs developed using PCN are portable across many different workstations, networks, parallel computers. The code portability aspect of PCN makes it suitable as an implementation system for Cluster-M.

PCN focuses on the notion of program composition and emphasizes the techniques of using combining forms to put individual components (blocks, procedures, modules) together. This encourages reuse of parallel code since a single combining form can be used to develop many different parallel programs. In addition, this facilitates reuse of sequential code and simplifies development, debugging and optimization, by exposing basic structure of parallel programs. PCN

provides a core set of three primitive composition operators: parallel, sequential, and choice composition, represented by ||, ; and ? respectively. It is a simple, high-level programming language with C-like syntax. More sophisticated combining forms can be implemented as user-defined extensions to this core notation. Such extentions are referred to as templates or user-defined composition operators. Program development, both with the core notation and the templates is supported by a portable toolkit. The three main components of the PCN system are illustrated in Figure 9.

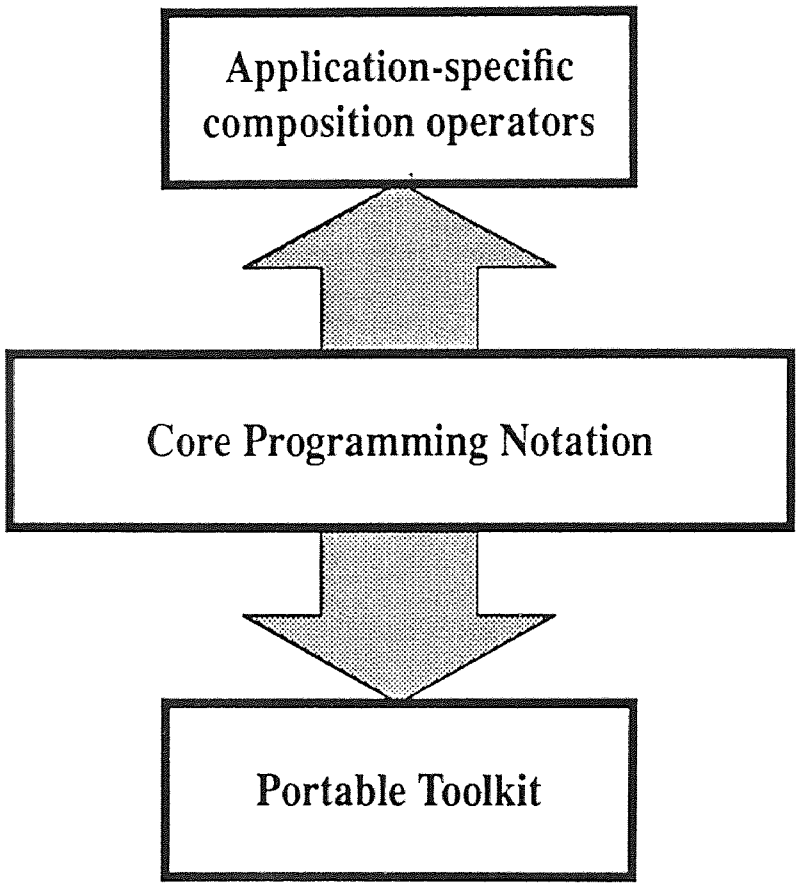


Figure 9. PCN System Structure

3.2 PCN Cluster-M Constructs

The seven Cluster-M constructs are implemented in PCN as follows:

```
/* 1. Makes given elements into one cluster */
CMAKE(LVL, ELEMENTS, x)
{ || MIN(ELEMENTS, n),
/* n is the smallest number in ELEMENTS */
x = [LVL, n, ELEMENTS]
}


MIN(E, n)
{ ? E? = [m|E1]- >
    { ; MIN1(E1, m, min),
        n = min
    }
}


    MIN1(E1, m, min)
{ ? E1? = [h|E2]- >
    { ;
        {?h < m- > m1 = h,
        default- > m1 = m
        } ,
```

$MIN1(E2, m1, min)$

},

$default- > min = m$

}

/* 2. Yields an element of the cluster */

$CELEMENT(x, j, e)$

{ ? $x? = [LVL|x1]- >$

{ ? $x1? = [n|x2]- >$

$CELEMENT1(x2, j, e)$

},

$default- > e = []$

}

$CELEMENT1(x, j, e)$

{ ? $j > 1- >$

{ ? $x? = [h|x1]- >$

$CELEMENT1(x1, j - 1, e)$,

},

$default- > e = x$

}

/* 3. Yields the size of the cluster */

$CSIZE(x, s)$

$\{ ?\ x? = [\_, \_, x2] -> CSIZE1(x2, 0, s),$
$\quad default \quad -> s = 0$
$\}$

$CSIZE1(x, acc, s)$

$\{ ?\ x? = [\_|x1] -> CSIZE1(x1, acc + 1, s),$
$\quad default \quad -> s = acc$
$\}$

/* 4. Merges cluster x and y */
$CMERGE(x, y, ELEMENTS, z)$

$\{ ?\ [LVL\_x|x1], y? = [LVL\_y|y1] ->$
$\quad \{ ?\ x1? = [nx|x2], y1? = [ny|y2] ->$
$\quad\quad \{ ||\ MIN(nx, ny, min),$
$\quad\quad z = [LVL_x + 1, min, ELEMENTS]$
$\quad\quad \},$
$\quad \},$
$\quad default -> z = []$
$\}$

$MIN(nx, ny, min)$

$\{ ?\ ny >= nx -> min = nx,$

$\quad default \quad -> min = ny$

$\}$

/* 5. Does the Unary operation */

$CUN(*, x, i, e)$

$\{ ||\ CELEMENT(x, i, e1),$

$\quad e = *(e1),$

$\}$

/* 6. Does the Binary operation */

$CBI(*, x, i, y, j, e)$

$\{ ||\ CELEMENT(x, i, e1),$

$\quad CELEMENT(y, j, e2),$

$\quad e = e1 * e2,$

$\}$

/* 7. Does the Split operation */

$CSPLIT(x, k, p, q)$

$\{\ CSIZE(x, s),$
$\quad \{?\ k > s- >$
$\quad\quad \{\ ||\ x = [LVL|x1],$
$\quad\quad\quad x1 = [n|x2],$
$\quad\quad\quad CSPLIT1(x2, k, p),$
$\quad\quad \}$
$\quad\quad CSPLIT2(x2, k, s - k, q),$
$\quad \}$
$\}$

$CSPLIT1(x, k, p)$

$\{?\ k > 0- >$
$\quad \{\ ||\ x = [h|x1]\ ,$
$\quad\quad p = [h|p1],$
$\quad\quad CSPLIT(x1, k - 1, p1),$
$\quad \}$
$\quad default- > p = []$
$\}$

$CSPLIT2(x, k, l, q)$
$\{?\ k > 0- >$
$\quad \{\ ||\ x = [h|x1]\ ,$
$\quad\quad CSPLIT2(x1, k - 1, l, q),$
$\quad \}\ ,$

$default- > CSPLIT1(x, l, q)$

}

3.3 PCN Cluster-M Macros

Several operations are frequently encountered in designing parallel algorithms. Macros can be defined using basic Cluster-M constructs to represent such common operations. The utilization of macros in problem Specifications instead of using low-level constructs simplifies mapping of Specifications to Representations. The mapping of each defined macro is done for each system Representation only once. Whenever any defined macro is encountered in the problem Specification, the predetermined mapping for the architecture at hand is looked up from a Cluster-M macro mapping library. We next present several macros, their coding in terms of Cluster-M constructs and their PCN implementation:

3.3.1 Associative Binary Operation

Performing as associative binary operation on N elements ending up with one value as the result is a common operation in parallel applications. The Cluster-M Specification for input size = 8 is given in Figure 10. The resulting Specification is an inverted tree with input values each in a leaf cluster at level 1 and the result at root cluster at level $\log n + 1$. Using Cluster-M constructs, the macro ASSOC-BIN

applies associative binary operation $*$ to the N elements of input A and returns the resulting value as follows:

$ASSOC\_BIN(*, N, A)$

{ ; $LVL = 1$,
{ || op i over 1 to N
$CMAKE(LVL, i, A(i))$},
$k = \log N$,
{ || op LVL over 1 to K
$CMERGE(x, y, CBI(op, x, i, y, j, e))$
}

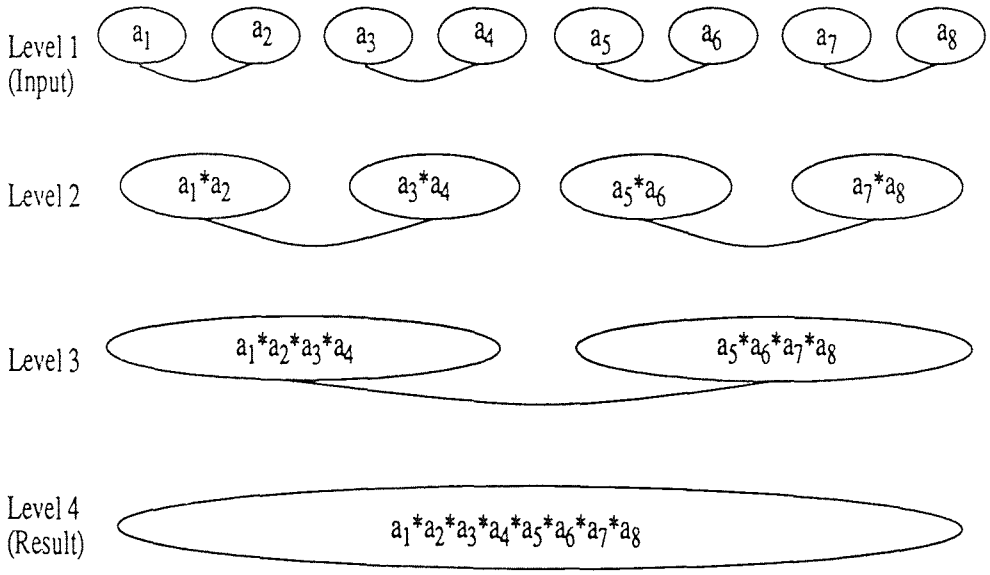


Figure 10. Cluster-M Specification of Associative Binary Macro.

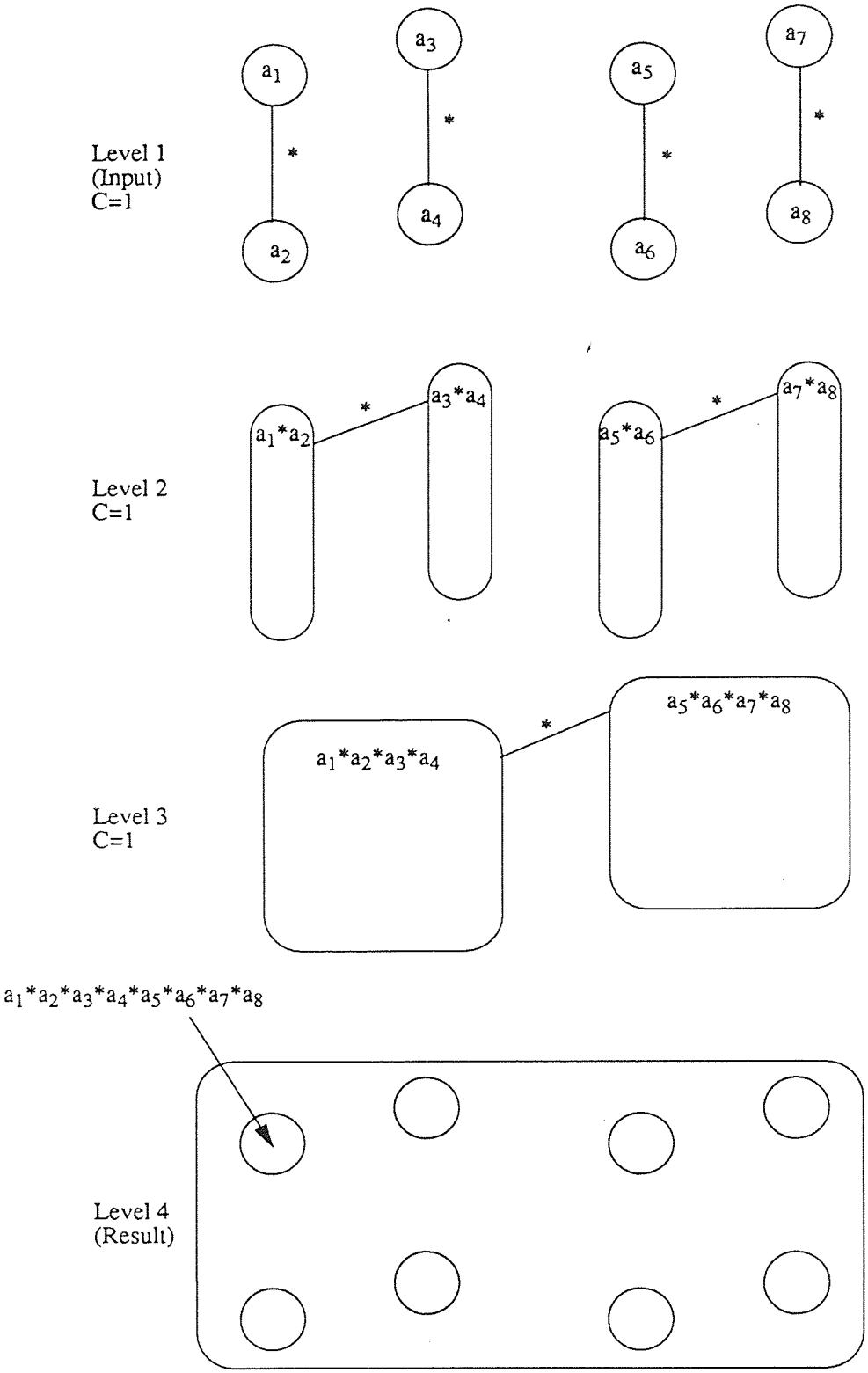


Figure 11. Mapping of Associative Binary Macro Onto An N-Cube of Size 8.

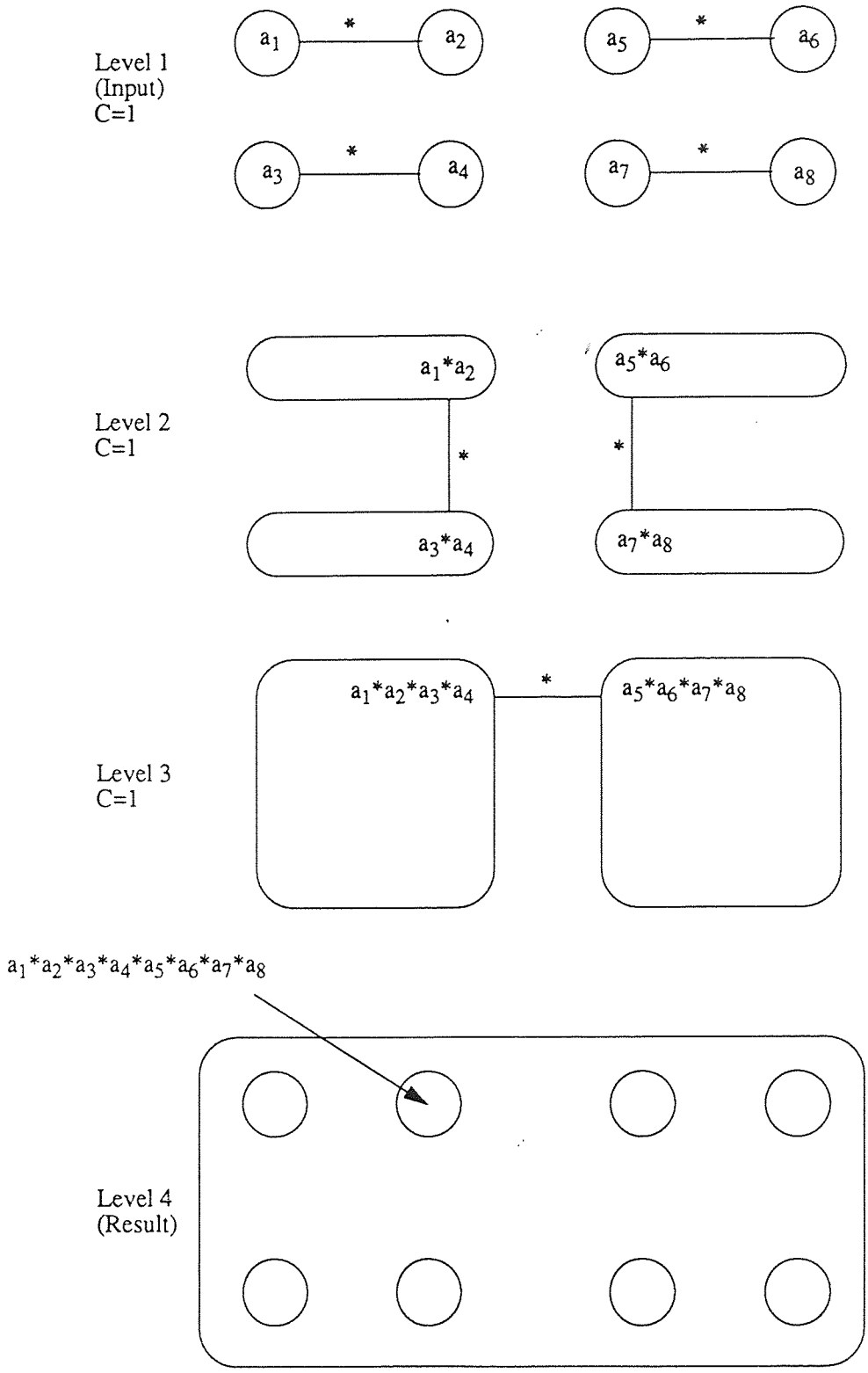


Figure 12. Mapping of Associative Binary Macro Onto A Mesh of Size 8.

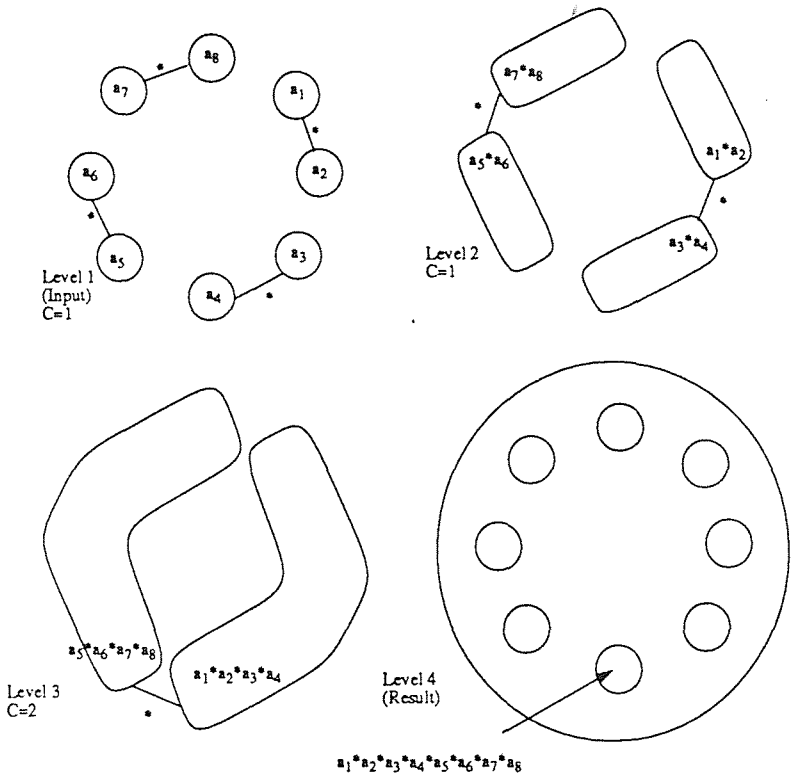


Figure 13. Mapping of Associative Binary Macro Onto A Ring of Size 8.

3.3.2 Vector Dot Product

As a representative example of vector operations(Vecops), we consider here the dot product of two vectors. The vector dot product of two n-element vectors A and B is defined as $d = \sum_{i=1}^{n}(a_i \cdot b_i)$. The cluster-M Specification for $n = 8$ is given in Figure 14.

The first level of clustering has each vector pair of vector elements a_i, b_i in adjacent clusters each containing one element. The clusters are merged by multiplying each two elements. Each two adjacent clusters are merged by adding their elements. This is continued till a single-cluster level is reached. This macro can be written in terms of Cluster-M constructs and the above ASSOC-BIN macro as follows:

$$MACRODOT\_PRODUCT(*, N, arrayA[i], arrayB[j])$$

$int\ arrayA[], arrayB[], i, j,$
$LVL = 1,$
{ || op over 1 to $2 * N - 1$
$CMAKE(LVL, i, arrayA[i]),$
$CMAKE(LVL, i + 1, arrayB[i])\}$,
{ || op i over 1 to $2 * N - 1$
$CMERGE(LVL, i, i + 1, CBI(op, x, i, y, j))\}$,
{ ; $ASSOC\_BIN(*, N, CELEMENT(LVL + 1, -, -))\}$
}

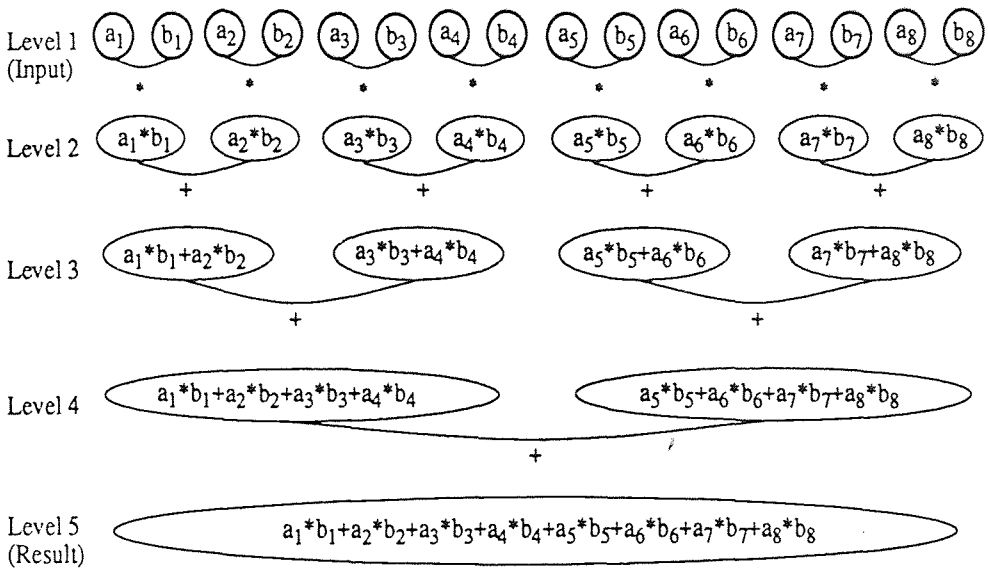


Figure 14. Cluster-M Specification of Dot Product Macro.

3.3.3 SIMD Data Parallel Operations

In this class of operations each operation is applied to all the input elements without any communication. In this case each operand is assigned one cluster in the problem Specification. The desired operation is applied to all clusters. The macro DATA-PAR applies operation $*$ to all N elements of input A, as follows:

$$MACRODATA\_PAR(*, N, A)$$

$\{$; $LVL = 1$, $FUNCT = A[i]$
$\{ \|$ op over 1 to N
$CMAKE(LVL, i, FUNCT)$
$\}$
$\}$

3.3.4 Broadcast Operation

This is a frequently encountered operation in parallel programs. One value is to be broadcast to all processors in the system.

The problem Specification for a macro that broadcasts one value 'a' from processor x to N recipient clusters or processors, can be written in terms of Cluster-M constructs as follows:

$BROADCAST(a, x, n)$

$\{ ; LVL = 1, i! = x$
$\quad || \text{ op i over 1 to N}$
$CMAKE(LVL, i, a)$
$\quad \}$
$\}$

The Specification of the broadcast operation for $N = 8$ and its mapping onto a completely connected system of size 8 is shown in Figure 15.

3.4 PCN Representation Algorithm

The following pseudo-code algorithm, $SYS - REP$, constructs the Cluster-M Representation of a connected system of N processors.

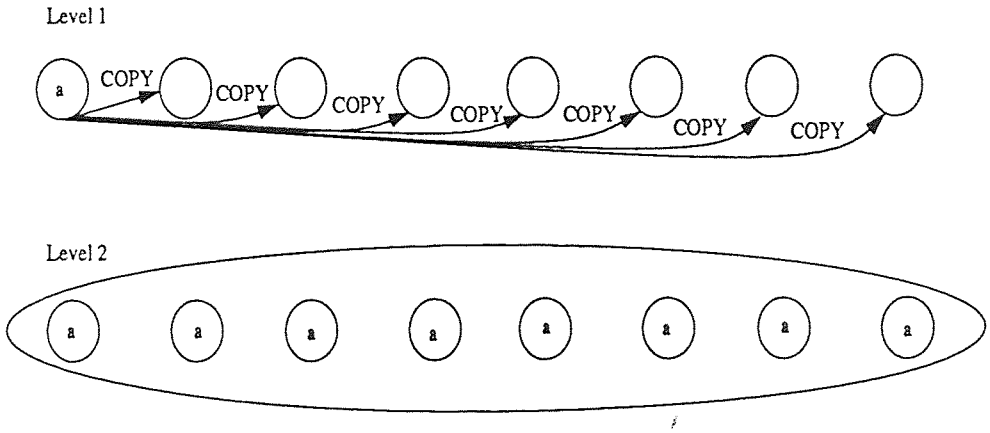


Figure 15. Cluster-M Specification of Broadcast Macro.

Initially, all clustering levels are empty. At clustering level 1, each system processor is in a cluster by itself. For each clustering level, the clique containing the lowest-numbered un-merged cluster, is obtained using procedure $CLIQUE$. The details of finding cliques is omitted (for any of several existing algorithms can be utilized). All clusters in the obtained clique are then merged into one cluster of the next clustering level using procedure $MERGE$. This is continued until all clusters of the current level are merged. The algorithm halts when a clustering level is reached which is comprised of one cluster with label 1.

PROCEDURE $SYS - REP(A)$

For all, i, LVL

begin

$C(LVL, i) = 0$

$PC(LVL, i) = \phi$

$CLUSTERS(LVL) = []$

$LVL = 1$ cluster level set to 1

end

For all processors i, $1 \leq i \leq N$

begin

$C(LVL, i) = i$

Each processor is in a cluster by itself at level 1

$PC(LVL, i) = [i]$

$CLUSTERS(LVL) = CLUSTERS(LVL) + i$

end

While $CLUSTERS(LVL) \neq [1]$ do

begin

For all $c \in CLUSTERS(LVL)$ starting with min(c) do

begin

For all $x, y \in CLIQUE(LVL, c)$ do

begin

$MERGE(LVL, x, y)$

end

end

$LVL = LVL + 1$

end

PROCEDURE $CLIQUE(LVL, c)$

begin

Find $CLIQUE$ such that $c \in CLIQUE$

and $\forall x, y \in CLIQUE$

$\exists A(PC(x, LVL), PC(y, LVL)) = 1$

end

PROCEDURE $MERGE(LVL, x, y)$

begin

$CLUSTERS(LVL{+}1) = CLUSTERS(LVL{+}1){+}\min(x, y)$

$PC(LVL + 1, \min(x, y)) = PC(x, LVL) + PC(y, LVL)$

For all p, $C(LVL, p) = x$ or y do

begin

$C(LVL + 1, p) = \min(x, y)$

end

end

The PCN version of the Cluster-M Representation algorithm is given in the Appendix.

CHAPTER 4
MAPPING SPECIFICATIONS TO REPRESENTATIONS

The most challenging task in the Cluster-M model is the mapping of the Specifications onto the fixed Cluster-M Representations of various architectures. Although in some cases this may appear simple, the mapping of certain Specifications may be non-trivial. For example, consider the associative binary operation example of the last chapter. We assume that it will take one time unit for a single communication along a link. Its mapping onto a 3-cube is shown Figure 16 and is straight forward. In step 1 two clusters each having one element are merged in one time unit. In step 2, two clusters each having two elements are merged in two time units. In step 3, two clusters each having four elements are merged in four time units into one cluster having 8 elements. So Mapping onto the 3-cube is done in 3 steps.

On the other hand, to map the same onto a binary tree of size 8 will lead to a greater time complexity since there are not enough communication channels available to support the communication request specified in the Cluster-M Specification. The complexity of the Specification onto the Ring and Mesh of size 8 is shown below;

Mapping onto Ring of size 8:

The Mapping onto Ring of size 8 will also be done in 3 steps but the time complexity increases. It will take 1 unit of time for step1,

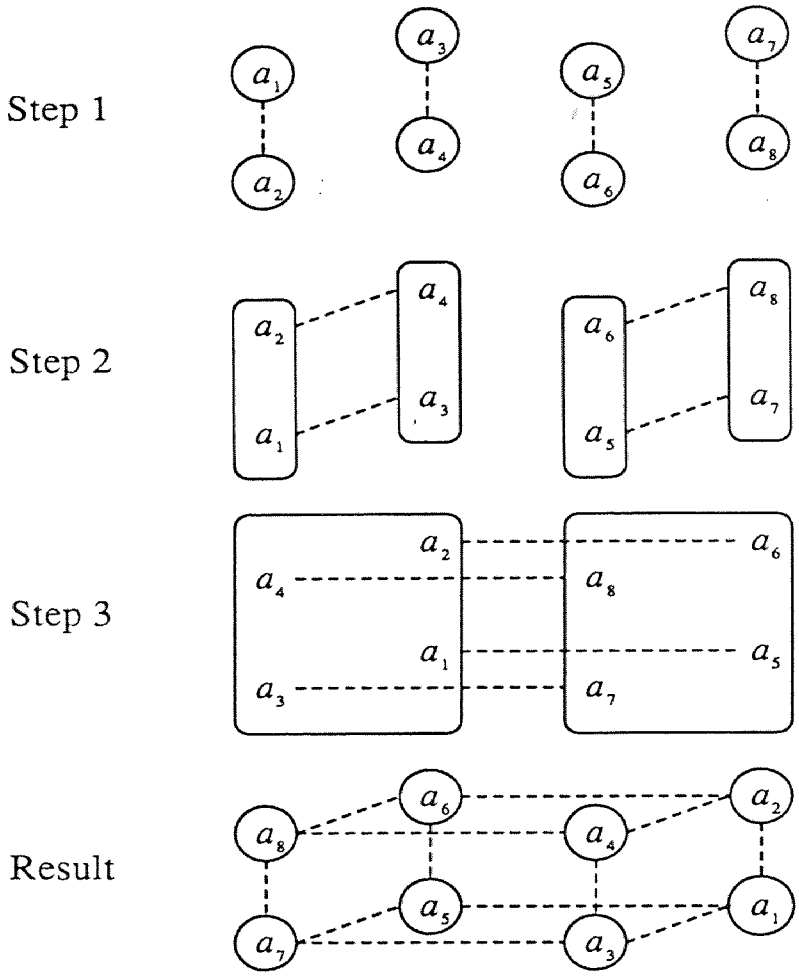


Figure 16. Mapping Onto N-Cube of Size 8

2 units of time for step2 and 4 units of time for step3.

Mapping onto Mesh of size 8:

The Mapping onto Mesh of size 8 will also be done in 3 steps but the time complexity differs from the above to mappings. Here it will take 1 unit of time for step1, 1 unit of time for step 2 and 2 units of time for step 3.

Similarly, there is going to be a slowdown if there are not enough processors in the Representation available as specified in the Specification. For example, the same problem described above, will take at least twice as much time if it is to be mapped on a Cluster-M Representation having half the number of processors. Mismatch of the number and structure of clustering in Cluster-M Specfication versus Cluster-M Representation may lead to significant slow performance. In the following section we present an efficient methodology for mapping an arbitrary Specification to Representation.

4.1 A Mapping Methodology

A good strategy for mapping of a parallel computing application onto a system of interconnected processors aims at maximizing the utilization of the available processing and communication resources, leading to faster execution times. This is traditionally accomplished by thorough analysis of the problem graph in terms of computation blocks

granularity and data dependencies between such blocks. The system parameters, namely processor power and interconnection topology, are also carefully analyzed. The mapping process then attempts to match each computation block with a system processor minimizing system communication overhead (i.e minimize the number of system communication hops for each data dependency in the problem).

The Cluster-M paradigm simplifies the mapping process by formulating the problem in the form of Cluster-M problem Specification emphasizing its computation and communication requirements independently from the target architecture. Similarly, the Cluster-M Representation of the system emphasizes the topology of the target multi-processor system. Once both, the Cluster-M problem Specification and system Representation are obtained the mapping process proceeds as follows:

Start from the root of Cluster-M specification. At level i, there are a number of clusters. Each cluster has a size K which is defined by the cumulative sum of the number of computations involved in all its nested subclusters. On the other hand, in Cluster-M representation, we have a collection of subclusters as part of a Cluster-M representation of a single connected system. We next look for a number of clusters in the representation to match the number of clusters at the ith level of the specification. Furthermore, we select the clusters such that the size of the corresponding pair matches. The details of this algorithm are beyond the scope of this thesis. For more information, see [13]. As part of the proposed algorithm, several graph theoretic

techniques have been used. In the next section, we give an example to illustrate the functionality of the mapping module.

4.2 An Example

In this section, we present a complete example to illustrate the Cluster-M mapping methodology presented above.

Figure 17 shows the mapping from a Cluster-M specification to representation. First of all, two clusters at the top level of specification are mapped onto two clusters of representation. The specification cluster of size 5 is mapped onto the representation cluster of the same size, however the specification cluster of size 4 has to be mapped onto the representation cluster of size 3 since this is the closest matching of sizes. Then the same procedure applies for the clusters at the lower level of specification. As shown in Figure 17 step 2, specification cluster a is mapped onto representation cluster H, which is a processor. In step 3, specification clusters b, e, f, g, h and i at specification 2 are mapped onto corresponding processors. Finaly in step 4, specification cluster c and d are both mapped onto processor F.

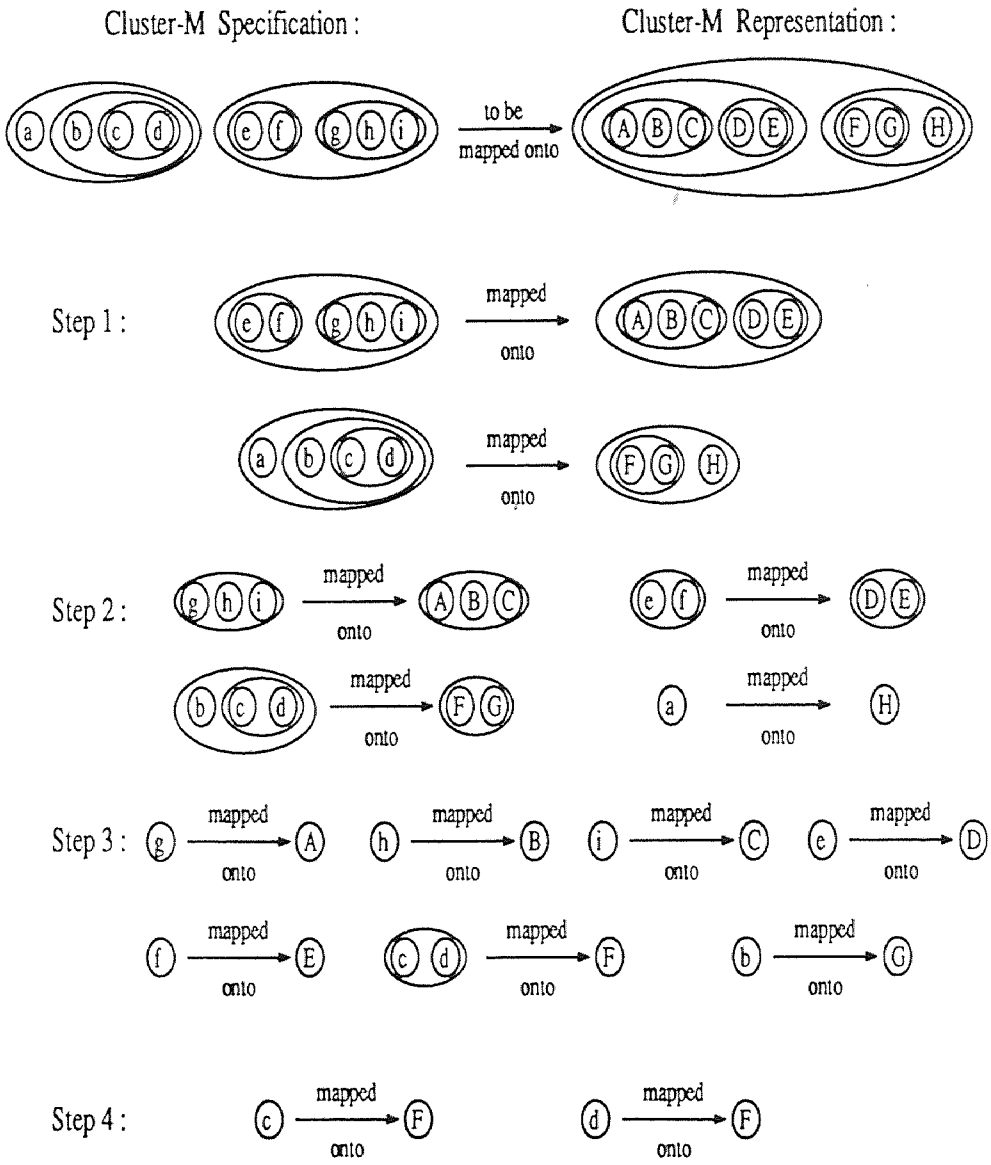


Figure 17. An Example For Mapping Algorithm

CHAPTER 5
CONCLUSION AND FUTURE RESEARCH

In this thesis we have described the PCN Implementation of the Cluster-M components which are the Cluster-M Specification and the Cluster-M Representation which includes the macros and the Representation algorithm. The constructs and the macros are executed on the SGI Workstation. The theoretical aspects of Mapping the Specification to the Representation is illustrated and the Implementation aspects is a part of ongoing research and will be discussed in [13].

APPENDIX

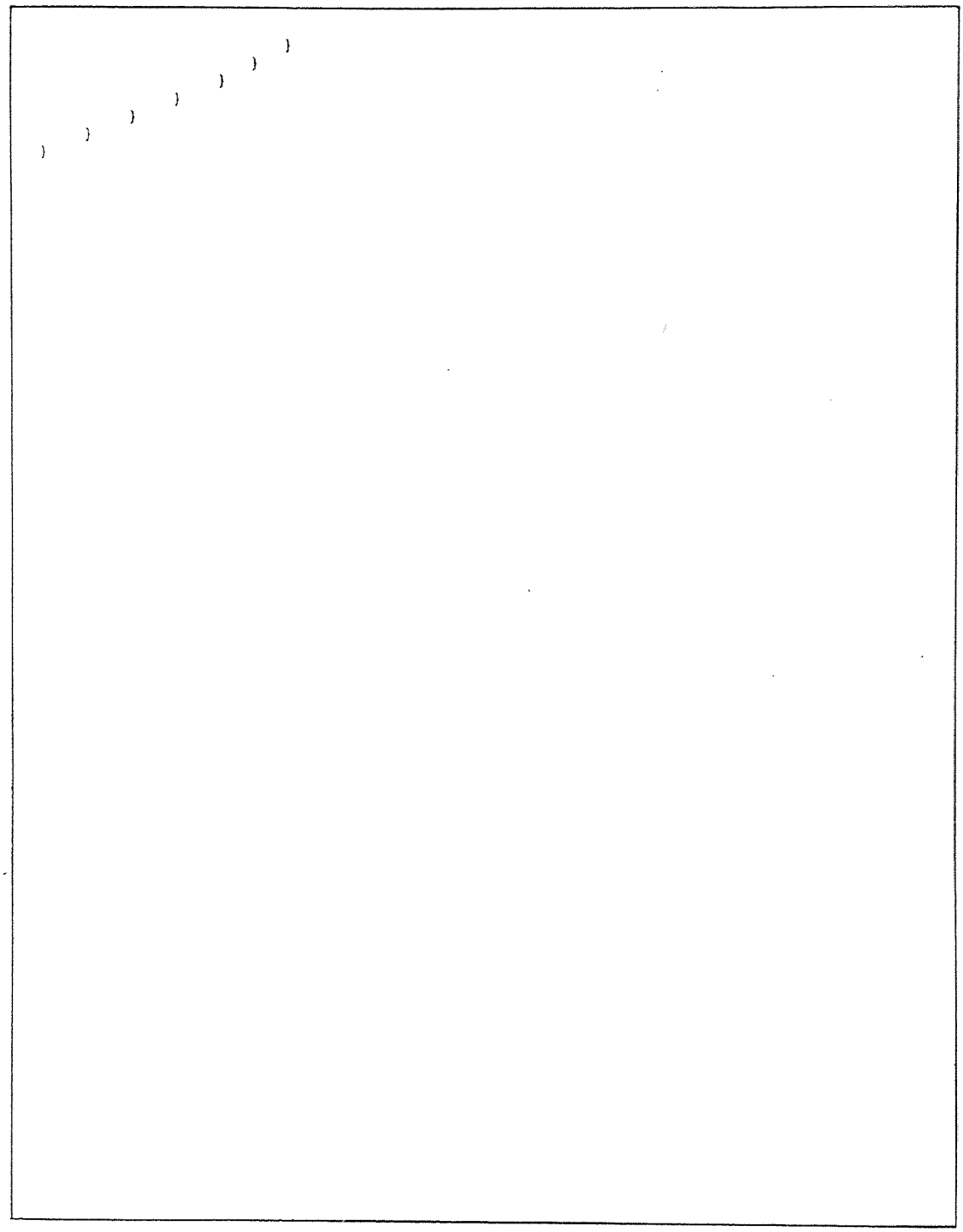
This appendix contains the PCN version of the Cluster-M Representation algorithm.

The PCN version of the Representation algorithm is given:

```
/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*          SYSTEM REPRESENTATION ALGORITHM USING PCN                */
/*                                                                   */
/*                                                                   */
/*********************************************************************/
DATA STRUCTURES USED IN THE ALGORITHM:

                Clusters : 2-D array of values 0 or 1 .
                           First dimension indicating the level and
                           second dimension indicating whether the
                           cluster numbered by that index is present
                           or not.
                Pr_cl    : 3-D array with 1st dimension indicating
                           level, second dimension index indicating
                           the cluster number and 3rd dimension index
                           indicating whether that processor is present
                           in that cluster or not . This array is also
                           binary.
                Member   : 2-D array with 1st dimension indicating
                           level, second dimension indicating processor
                           number and the value indicating to which
                           cluster  this processor belongs to.
                Clique   : 2-D array with first dimension
                           indicating level,second dimension indicating
                           cluster number of which this clique is,
                           third dimension indicating the processor
                           number and the value representing whether
                           the processor in the cluster is in the clique.
                           This array is also binary valued.


#define max 100  /* Maximum number of nodes in the system representation graph*/
/*
    INPUTS : None
    OUTPUTS : Number of graph nodes ,
              Adjacency Matrix of the graph nodes.
    Functionality : Reads the number of nodes and the adjacency matrix
                    from standard input.
*/

Input(N,adj)
int N;
int adj[max][max];
int i,j;
{;      scanf("%d",&N),     /* Read the number of nodes in the graph */
        {|| i over 1..N :: /* Initialise the adjacency matrix of graph *
                {|| j over 1..N ::
                        adj[i][j] = 0 )
        }
        {; i over 1..N ::  /* Read the adjacency matrix of the system graph */
                {; j over 1..N ::
                        scanf("%d",&adj[i][j])
                }
        }
)


/*
    INPUTS :    member , clusters,pr_cl,clique,N.
    OUTPUTS :   member , clusters,pr_cl,clique,N.
    Functionality : Initialises the variables used .
*/
```

```
Initialise(member,clusters,pr_cl,clique,N)
int member[max][max];
int pr_cl[max][max];
int clusters[max][max];
int clique[max][max][max];
int N;
int lvl,pr,cl,i,j,cq,k;
{;      lvl over 2..N ::   /* Initialising from level 2 to N      */
        {||     {|| pr over 1..N :: member[lvl][pr] = 0),
                {|| cl over 1..N :: clusters[lvl][cl] = 0),
                {||     i over 1..N  ::
                        {||     j over 1..N ::
                                pr_cl[lvl][cl][pr] = 0
                        }
                },
                {||     cq over 1..N ::
                        {||     k over 1..N ::
                                clique[lvl][cq][pr] = 0
                        }
                }
        },
        {|| cl over 1..N ::  /* Initialising for level 1 */
                {||     member[1][cl] = cl,
                        pr_cl[1][cl][cl] = 1,
                        clusters[1][cl] = 1,
                        clique[1][1][1] = 1
                }
        }
}


/*
   INPUTS :     clusters, lvl.
   OUTPUTS :    number.
   Functionality : Calculates the number of clusters at level lvl
                   using 2-D matrix clusters and returns this value
                   in number.
*/
no_of_clusters(clusters,lvl,number)
int lvl;
int clusters[max][max];
int number;
int cl;
{;      number := 0,
        {||     cl over 1..N ::
                {?      clusters[lvl][cl] == 1 ->
                                        number := number + 1
                }
        }
}


/*
   INPUTS :     lvl, cl, n_cl, member, pr_cl.
   OUTPUTS :    member, pr_cl.
   Functionality : Merges the clusters numbered cl and n_cl into
                   cl and accordingly updates the array member and
                   pr_cl.
*/
merge(lvl,cl,n_cl,member,pr_cl)
int lvl,cl,n_cl;
int member[max][max];
int pr_cl[max][max][max];
{||     pr over 1..N ::
        {?      member[lvl][pr] == n_cl -> /* if a processor is a member of n_cl
```

```
,
                                                    make it a member of cl    */
                        {||     member[lvl+1][pr] = cl,
                                pr_cl[lvl+1][cl][pr] =1
                        }
        }
}


/*
   INPUTS :     lvl, clusters, member.
   OUTPUTS :    None.
   Functionality : Outputs the clusters in each level to standard
                   output.
*/
Output(lvl,clusters,member)
int lvl,clusters[max][max];
int member[max][max][max];
int i,cl,pr;
{;      printf("lvl : %d ",lvl),
        {;      cl over 1..N ::
                {?      clusters[lvl][cl] ==1 ->
                        {; printf("("),
                           printf("%d :",cl),
                           {; pr over 1..N ::
                                {?  member[lvl][pr] == cl->
                                        printf("%d ",pr)
                                },
                                printf(")"
                           }
                        }
                },
                printf("\n");
        }
}


/*
   INPUTS :             lvl, c, x, clique, pr_cl, adj.
   OUTPUTS :            flag.
   Functionality :      Checks if clusters numbered by c and x form a clique
                        in the system representation graph and returns the flag
value
                        as 1 if they form clique and 0 otherwise.
*/
in_clique(lvl,c,x,clique,pr_cl,adj,flag)
int lvl,c,x,clique[max][max][max];
int pr_cl[max][max][max],adj[max][max];
int flag,y,pc_x,pc_y;
{;  flag =1,
    {; y over 1..N ::  /* for all the processors in the clique of cluster c */
        {?  clique[lvl][c][y] == 1->
            {? flag == 1 ->
                {; flag = 0,
                   {|| pc_x over 1..N :: /* for all the processors in the
                                            cluster x     */
                      {? pr_cl[lvl][x][pc_x] ==1 ->
                         {|| pc_y over 1..N :: /* for all the processors in
                                                  the clusters that are in
                                                  clique formed by c  */
                             {? pr_cl[lvl][y][pc_y] == 1->
                               ? adj[pc_x][pc_y] == 1->  /* if they are adjacent
*/
                                        flag := 1
                             }
                          }
```



REFERENCES

[1] N.Carriero, D.Gelernter, and J. Leichter. January 1986. "Distributed Data Structures in Linda." *Proceedings of the Thirteenth ACM Symposium on Principles of Programming Languages.*

[2] G.Agha. 1986. *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press, Cambridge, Mass.,

[3] Kai Hwang, Faye A. Briggs. *Computer Architecture and parallel processing.* McGraw Hill International Series.

[4] Ian Foster, Steven Tuecke. *Parallel Programming with PCN.* Argonne National Laboratory, University of Chicago.

[5] M.Chandy and S.Taylor. 1991 *An Introduction to Parallel Programming* Jones and Barlett.

[6] Mary. M. Eshaghian and Muhammad E. Shaaban. April 1993 " Cluster-M Parallel Programming Paradigm." *International Parallel Processing Symposium.*

[7] Mary. M. Eshaghian and R. F. Freund. March 1992. "Cluster-M Paradigms for High-Order Heterogeneous Procedural Specification", *Heterogeneous Workshop.*

[8] Mary. M. Eshaghian July 1991. "Parallel Algorithms for Image Processing on OMC." *IEEE Transactions on Computers: Vol.40*, No.7.

[9] Leah. H. Jamieson. 1987 *Characterizing Parallel Algorithms.* MIT Press Series in Scientific Computation.

[10] A. Khokhar, V. K. Prasanna, M. Shaaban, C. Wang March 1992. "Heterogeneous Supercomputing: Problems and Issues." *Proceedings Workshop on Heterogeneous Processing*, pp 3-12.

[11] George S. Almasi and Alan Gottlieb. 1989. *Highly Parallel Computing.* The Benjamin/Cummings Publishing Company, Inc.

[12] Joseph JaJa. 1992. *An Introduction to Parallel Algorithms.* Addison-Wesley Publishing Company.

[13] Ajitha Gadangi *Implementation of Mapping Module.* Thesis to appear, New Jersey Institute of Technology.