

**OREGAMI: Tools for Mapping
Parallel Computations to
Parallel Architectures**

Virginia Lo, Sanjay Rajopadhye, Samik Gupta,
David Keldsen, Moataz Mohamed, Bill Nitzberg,
Jan Telle, and Xiaoxiong Zhong

CIS-TR-89-18a
Revised April 1, 1992

OREGAMI: Tools for Mapping Parallel Computations to Parallel Architectures*

Virginia M. Lo[†], Sanjay Rajopadhye[‡]
Samik Gupta, David Keldsen, Moataz A. Mohamed[§],
Bill Nitzberg, Jan Arne Telle, and Xiaoxiong Zhong
Dept. of Computer and Information Science
University of Oregon
Eugene, Oregon 97403-1202
503-686-4408
lo@cs.uoregon.edu

Keywords: mapping, routing, embedding, task assignment,
regular parallel computations, parallel programming environments

Abstract

The OREGAMI project involves the design, implementation, and testing of algorithms for mapping parallel computations to message-passing parallel architectures. OREGAMI addresses the mapping problem by exploiting regularity and by allowing the user to guide and evaluate mapping decisions made by OREGAMI's efficient combinatorial mapping algorithms. OREGAMI's approach to mapping is based on a new graph theoretic model of parallel computation called the Temporal Communication Graph. The OREGAMI software tools includes three components: (1) LaRCS is a graph description language which allows the user to describe regularity in the communication topology as well as the temporal communication behavior (the pattern of message-passing over time). (2) MAPPER is our library of mapping algorithms which utilize information provided by LaRCS to perform contraction, embedding, and routing. (3) METRICS is an interactive graphics tool for display and analysis of mappings. This paper gives an overview of the OREGAMI project, the software tools, and OREGAMI's mapping algorithms.

*This research was sponsored by Oregon Advanced Computing Institute, a consortium of academic, industrial, and government agencies in the state of Oregon.

[†]Partially supported by NSF grant CCR-8808532

[‡]Partially supported by NSF grant MIP-8802454

[§]Currently with MIPS Computer Systems, Sunnyvale, CA.

1 Introduction

The *mapping problem* in message-passing parallel processors involves the assignment of tasks in a parallel computation to processors and the routing of inter-task messages along the links of the interconnection network. Most commercial parallel processing systems today rely on manual task assignment by the programmer and message routing that does not utilize information about the communication patterns of the computation. The goal of our research is *automatic* and *guided* mapping of parallel computations to parallel architectures in order to achieve portability and to improve performance.

The OREGAMI¹ project involves the design, implementation, and testing of mapping algorithms. OREGAMI's approach to mapping is based on (1) its use of a new graph theoretic model of parallel computation which we call the Temporal Communication Graph, and (2) its exploitation of regularity found in the structure of parallel computations and of the target architecture.

We are concerned with the mapping of parallel computations which are designed by the programmer as a collection of communicating parallel processes. We note that many practical parallel computations are characterized by regular communication patterns. This regularity occurs both in the topological communication structure of the computation (which tasks send messages to whom) and in the temporal communication behavior exhibited by the computation (the patterns of message-passing phases over time). Furthermore, the programmer has explicit knowledge of this regularity because it forms the basis of the logical design of her/his computation. While the general mapping problem in its many formulations is known to be NP-hard, it has been shown that restriction of the problem to regular structures can yield optimal solutions or good suboptimal heuristics for mapping.

OREGAMI addresses the mapping problem by exploiting regularity whenever possible, and by allowing the user to guide and evaluate mapping decisions made by OREGAMI's mapping algorithms. Our mapping system balances the user's knowledge and intuition with the computational power of efficient combinatorial mapping algorithms.

The contributions of the OREGAMI project include:

- **The TCG model of parallel computation and the LaRCS graph description language.** The TCG model is a new graph theoretic model of parallel computation designed for the purpose of mapping. The TCG can be seen as an augmented version of

¹For University of OREGON's techniques for elegant symmetric contractions which resemble the art of oriGAMI paper folding.

Lamport's process-time graphs. The TCG integrates the two dominant models currently in use in the areas of mapping and scheduling: the static task graph and the DAG. LaRCS (Language for Regular Communication Structures) is a graph description language which provides the user with the ability to describe the parallel computation's TCG in a natural and compact representation. LaRCS provides the capability to identify logically synchronous *phases* of communication, and to describe the temporal behavior of a parallel computation in terms of these phases in a notation called *phase expressions*.

- **Mapping algorithms which exploit regularity to yield high performance mappings.** Our mapping algorithms utilize the information provided by LaRCS to achieve mappings that are an improvement over uninformed mapping in two ways: the mapping algorithms themselves are efficient and the resultant mappings are optimal or near optimal based on a variety of standard performance metrics.
- **The OREGAMI software tools.** The OREGAMI tools include the LaRCS compiler; MAPPER, a library of mapping algorithms; and METRICS an interactive graphics tool for display and analysis of OREGAMI mappings.

OREGAMI is designed for use as a front-end mapping tool in conjunction with parallel programming languages that support explicit message-passing, such as OCCAM, C*, Dino [34], Par [13], and C and Fortran with communication extensions. The underlying architecture is assumed to consist of homogeneous processors connected by some regular network topology, with current focus on the hypercube, mesh, and deBruijn topologies. Routing technologies supported by OREGAMI include store-and-forward, virtual cut-through, and wormhole routing. Systems such as the Intel iWarp, Intel iPSC machines, NCUBE hypercubes, and INMOS Transputer are candidates for use with OREGAMI. Note, however, that OREGAMI is a front-end mapping interface and generates symbolic mapping directives only. Development of back-end software to transform OREGAMI mapping directives to architecture dependent code is beyond the current scope of our work.

This paper provides an overview of the OREGAMI project and software tools. Section 2 discusses the formal foundations underlying OREGAMI's approach to mapping and related research. Section 3 describes the components of OREGAMI: LaRCS, MAPPER, and METRICS, and traces its operation with an illustrative example. Section 4 briefly illustrates how the individual mapping algorithms that we have developed for OREGAMI exploit regularity. Section 5 discusses areas of on-going and future work on this project.

2 Formal Foundations and Related Research

2.1 Formal Foundations: the Temporal Communication Graph

The foundations for the OREGAMI system lie in a new graph theoretic model of parallel computation we have developed called the Temporal Communication Graph and its ability to capture regularity for purposes of mapping. Both the TCG and its forerunner, the classic static task graph of Stone [42], are designed for systems in which the programmer designs his or her program as a set of parallel processes that communicate through explicit message passing. The identity of all of the processes is known at compile time, and all processes exist throughout the lifetime of the parallel computation.

The TCG was designed to enrich the static task graph model and to provide a means for describing regularity in the structure of the parallel computation for the purpose of mapping. Specifically,

- **The TCG integrates three important graph theoretic models of parallel computation: Lamport's process-time graphs, the static task graph, and the DAG.** The TCG combines the two predominant models currently in use in the areas of mapping, task assignment, partitioning, and scheduling: the static task graph and the precedence-constrained DAG. The integration of these two models enables a wide spectrum of algorithms to be used for mapping and scheduling which could not otherwise be invoked because of incompatibilities in the underlying graph theoretic models. In addition, the compatibility of the TCG with Lamport's process-time graphs makes it useful as a unified abstraction in parallel programming environments for program development, debugging, and performance evaluation as well as for mapping and scheduling.
- **The ability of the TCG and the LaRCS graph description language to describe *regularity* facilitates the development and use of specialized mapping algorithms which exploit regularity to yield improved performance.** The TCG and LaRCS are capable of representing regularity both in the communication topology and in the temporal patterns of message passing over time.

2.1.1 Informal Description of the TCG

In this section, we give an intuitive description of the TCG. A formal definition of the TCG is given in [24] which also defines the formal semantics of LaRCS in terms of the TCG. Consider each individual process comprising the parallel computation. The activity of a given process

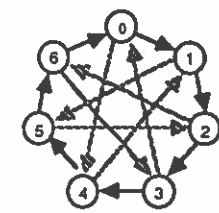
p_i can be seen as a sequence of atomic events, where each event is either a computation event or a communication event (sending a message/receiving a message). The TCG is a DAG in which each atomic event (compute, send, or receive) is represented as a node. The sequence of atomic events on p_i is represented as a linear chain of nodes, with directed edges indicating the precedence relationship between the events. A message-passing event from process p_i to process p_j is represented with a directed edge from the send-event node on p_i to the corresponding receive-event node on p_j .

The TCG can thus be seen as an unrolling of the static task graph over time. Conversely, the projection of the TCG along the time axis yields the static task graph model. Weights associated with the nodes and edges can be used to represent computation and communication costs, respectively. Note that the TCG also can be viewed as a graph theoretic representation of Lamport's process-time diagrams [21] augmented with weights and colors. The coloring of the Lamport process-time graph is described in [24] and involves the identification of logically synchronous communication and computation phases as described in the next section.

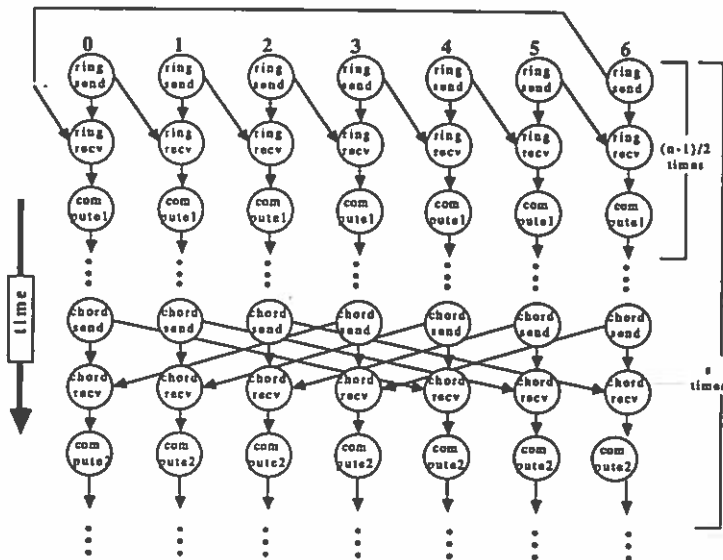
Figure 1 shows the TCG for a parallel algorithm for the *n-body* problem which was designed for the Cosmic Cube [38]. This algorithm will be used as an example later in the paper to illustrate the use of the OREGAMI mapping tools.

As discussed earlier, the TCG can be seen as a hybrid of the two predominant models of parallel computation: the static task graph model of Stone [42], and the precedence-constrained (DAG) model [30] used in multiprocessor scheduling and in the parallelization of sequential code. Task assignment and scheduling research utilizing these two models has more or less followed disjoint paths over the past two decades, in that techniques and algorithms developed for one model have not been applicable to the other. The TCG model is compatible with both of these existing models. Thus algorithms for static task assignment such as [8] [6] [23], [36] and scheduling algorithms for precedence-constrained graphs such as [30] [9] can be applied to the TCG model. The contribution of the TCG is that it augments these two models with the ability to explicitly capture regularity, allowing the development of specialized mapping and scheduling algorithms to exploit this regularity.

The TCG is also capable of modeling computations of arbitrary granularity, characterized by irregular and asynchronous communication. We note that the TCG does not model non-deterministic computations and dynamically spawned tasks. More information about the TCG model and its use in parallel programming environments can be found in [24].



→ ring
 → chordal



(a) 7-body static graph

(b) TCG of 7-body

The n -body problem requires determining the equilibrium of n bodies in space (where n is odd) under the action of a (gravitational, electrostatic, etc.) field. This is done iteratively by computing the net force exerted on each body by the others (given their "current" position), updating its location based on this force, and repeating this until the forces are as close to zero as desired. The parallel algorithm presented by Seitz uses Newton's third law of motion to avoid duplication of effort in the force computation. It consists of n identical tasks, each one responsible for one body. The tasks are arranged in a ring and pass information about their accumulated forces to its neighbor around the ring. After $(n - 1)/2$ steps, each task will have received information from half of its predecessors around the ring. Each task then acquires information about the remaining bodies by receiving a message from its chordal neighbor halfway around the ring. This is repeated to the desired degree of accuracy.

Figure 1: Temporal communication graph for the n -body algorithm

2.1.2 Describing Regularity in the TCG

We note that many practical parallel algorithms involve one or more *phases* of communication. These phases are often characterized by regularity in the communication topology, i.e., the static task graph is a known graph structure such as a mesh, a tree, etc. In addition, these algorithms also exhibit regularity in the temporal communication behavior (the patterns of message passing phases that are active over time).

A *compute phase* corresponds to a set of nodes in the TCG (compute events) that are involved in *logically synchronous* computation. A *communication phase* corresponds to a set of edges (sender/receiver pairs) in the TCG that are involved in *logically synchronous* communication. By *logically synchronous* we mean that at run time the activities occur simultaneously from the viewpoint of the programmer, i.e. from the logical structural design of the algorithm.²

These phases are identified using a node labeling scheme for each process. Each communication phase can then be described by a *communication function* whose domain and range are process node labels. For example, a ring of communicating processes can be described as ring: forall i in $0..n-1 \Rightarrow i+1 \bmod n$ where we assume the processes are labeled sequentially from 0 to $n-1$. In LaRCS, a notation called *phase expressions* is used to describe the temporal behavior of the parallel computation in terms of its compute phases and communication phases. In addition, LaRCS allows the user to describe families of regular graphs in a parameterized notation whose size is independent of the size of the task graph. The LaRCS description of the *n-body algorithm* and further details about LaRCS are described in the next section.

2.2 Related Research

As an integrated set of tools for mapping, OREGAMI is similar in many respects to Francine Berman's Prep-P System [4] [6]. Both Prep-P and OREGAMI provide a graph description language for describing the communication structure of the parallel computation. Prep-P's GDL language is based on the static task graph model of parallel computation and is embedded within the system's parallel programming language XX. LaRCS is based on our TCG model, which augments the static task graph with temporal information. The LaRCS specification is independent of any specific parallel programming language and is coded separately by the programmer. Prep-P is a fully integrated system: Prep-P mappings are targeted for the CHiP reconfigurable parallel architecture [39] and Prep-P currently generates code that runs on the CHiP simulator known

²In reality, when the program executes, the timing of logically synchronous activities may not be synchronous with respect to real time, due to effects such as the hardware characteristics of the execution environment and the multiplexing of processes on the processors.

as Poker. OREGAMI also is related to the TMAP mapping tool, a component of the TIPS transputer-based interactive parallelizing system [45]. TMAP is an adaptation of Prep-P for the transputer architecture. As discussed in the introduction, OREGAMI is currently a front-end mapping tool and only generates symbolic mapping directives.

OREGAMI is also related to a number of mapping and scheduling systems that utilize the classic DAG model of parallel computation such as CODE/ROPE [10], TaskGrapher [14], Polychronopoulos [31] and Sakar [37]. These systems differ from OREGAMI, Prep-P, and TMAP because they are designed for the purpose of parallelizing sequential code.

The OREGAMI mapping algorithms take the approach of many researchers who have attacked the mapping problem by exploiting the regularity found in the computation graph and/or the interconnection network. A large body of theoretical work on graph embeddings has yielded 1:1 mappings tailored for specific regular graphs; some of the more recent results are given in [33]. Many of these algorithms have or will be included in the OREGAMI library. Edge grammars [5] use formal language techniques to address the contraction of families of task graphs. Stone and Bokhari [42], [8] use a variety of graph theoretic algorithms to address task assignment for structures including trees, chains, and arbitrary task graphs. Work in the area of systolic arrays has yielded elegant mapping techniques for computations whose data dependencies can be expressed as affine recurrences [32] [12]. Related work in the area of application-dependent routing includes [19] and [7]. Our mapping algorithms build on these foundations and utilize techniques from group theory, graph theory, coding theory, and linear algebra. Other approaches to the mapping problem include search algorithms, linear programming, and clustering algorithms. These latter techniques typically do not exploit the regularity of the task graph.

The LaRCS language bears similarities to a number of graph description languages or configuration languages which have been developed for a variety of purposes in the area of parallel and distributed computing. These include formal approaches such as edge grammars [5] and graph grammars [20], [2], as well as more practical languages such as GDL [4], CONIC [28], GARP [20], and ParaGraph [3]. Of these, LaRCS, edge grammars, and GDL were designed specifically to address the mapping problem. LaRCS is unique in its ability to describe the temporal behavior of parallel computations. A primitive form of phase expressions was introduced by [29].

From the viewpoint of support environments for parallel programming, OREGAMI belongs to the family of systems which take a process-oriented view of parallel computation based on explicit message-passing. Examples of such systems include Prep-P [4, 6], Poker [39, 41], ORCA [40, 16], the Parallel Programming Environments Project [3], and TIPS [45]. OREGAMI and Prep-P focus on the mapping problem. The other projects address the broader issues of program design and development.

3 OREGAMI System Overview

This section describes the OREGAMI software tools and traces the use of these tools using the *n-body algorithm* as an example. Figure 2 illustrates the three components of the OREGAMI system.

The user first describes his/her parallel computation in LaRCS, by specifying (a) the static structure of the parallel computation graph using a node labeling scheme and simple communication functions, and (b) the temporal communication behavior of the computation using *phase expressions* (notationally similar to regular expressions). The LaRCS specification is program-independent, i.e., it can be used in conjunction with a variety of parallel programming languages to provide information about regularity to be found in the communication structure of the computation.

The LaRCS compiler translates the LaRCS code into an intermediate representation (an abstract syntax tree). The intermediate code is translated into the form needed for specific mapping algorithms, using OREGAMI utility functions to generate the desired data structures for each algorithm, such as the TCG, the static task graph, static task graphs corresponding to each communication phase, the individual communication functions, etc. The information is used by the MAPPER and METRICS software to perform mapping and to display and analyze the mapping, respectively.

The input to MAPPER includes the LaRCS description of the computation and a description of the target architecture which consists of the name plus parameters (e.g. hypercube of dimension 4). MAPPER examines any regular properties specified in the LaRCS code and uses one or more algorithms to do the mapping. These algorithms can be invoked automatically or through user selection. Many parallel computations have well known communication structures (such as trees, meshes, etc.) and the programmer may simply state this. OREGAMI has a library of “canned” mappings for such computations, and generating it simply involves a lookup in a library. Other algorithms may have a regular structure of a particular kind (such as node-symmetric). For such computations, OREGAMI uses specialized algorithms which are often extremely efficient. If no regularity can be exploited, efficient polynomial time algorithms are used to perform the mapping in three steps – contraction, embedding, and routing.

Finally, the user may inspect the mapping using METRICS. METRICS is an interactive graphics tool which displays the mapping along with a range of performance metrics reflecting load balancing, communication contention, and communication overhead. METRICS also allows the user to focus on specific processors or links and provides the opportunity for manual modification of the mapping.

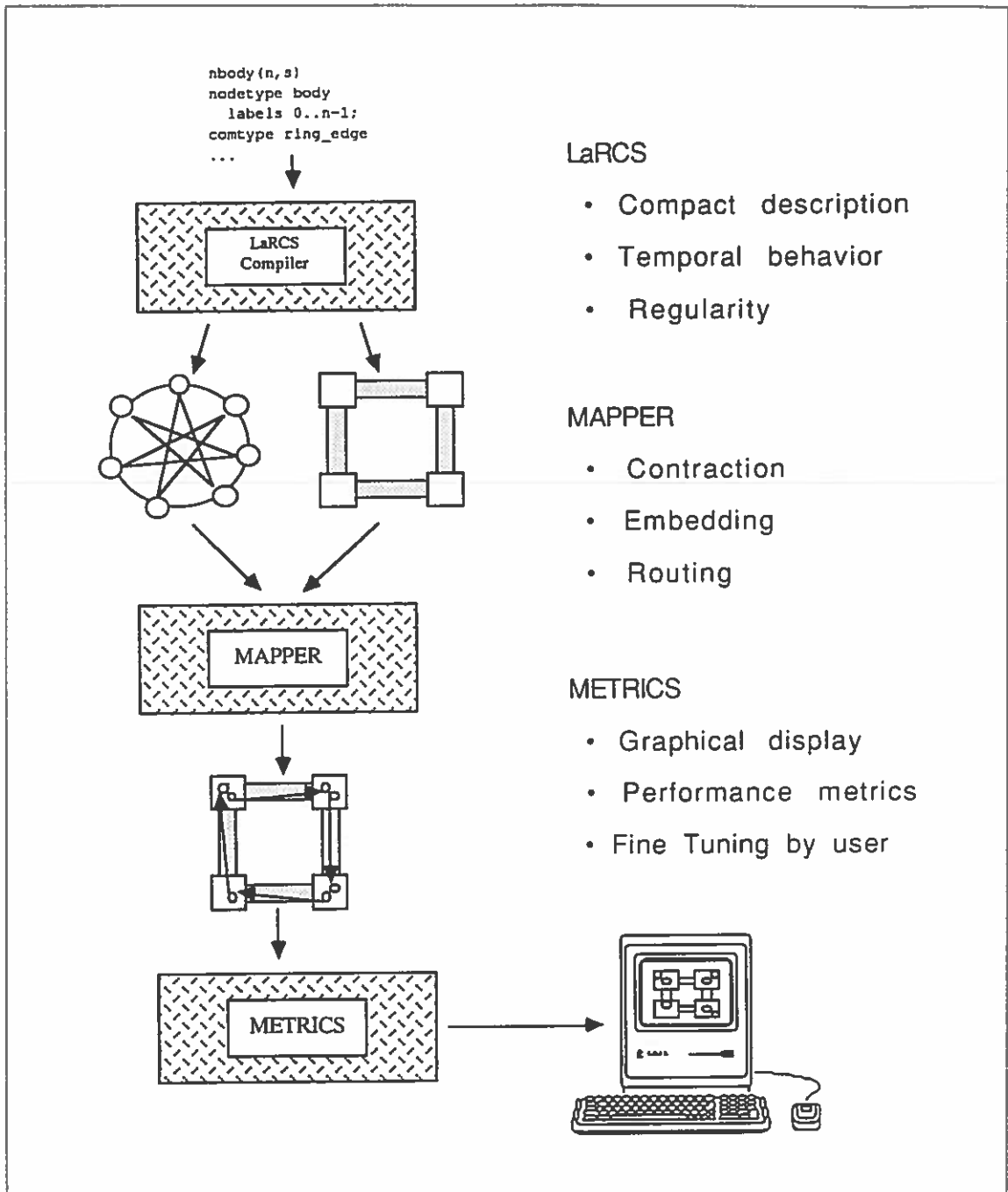


Figure 2: OREGAMI System Overview

The OREGAMI software tools are implemented in C for Sun workstations using Xlib, XtIntrinsic and the MIT Athena widget set. A new *topology* widget has been defined to display and manipulate various architecture types and mappings to them.

OREGAMI has been tested with a wide variety of parallel algorithms including several algorithms for matrix multiplication, fast Fourier transform, topological sort, divide and conquer using binary tree, divide and conquer using binomial tree, simulated annealing, Jacobi iterative method for solving Laplace equations on a rectangle, successive-over-relaxation iterative method, perfect broadcast distributed voting, numeric integration, distributed dot product, five point difference operation, Gaussian elimination with partial pivoting, matrix row rotation, and the simplex algorithm.

The performance of the OREGAMI mapping algorithms was evaluated through analytic proofs and through simulations. Simulations were performed using a library of C programs we have developed which generate task graphs derived from actual parallel programs as well as random task graphs and a non-backtracking branch and bound program to compute optimal mappings.

3.1 LaRCS

A LaRCS program consists of the following major components: (a) the LaRCS `nodetype` declaration which describes the processes, (b) the LaRCS `comtype` and `comphase` declarations which are used as templates to describe the communication structure of the parallel program, and (c) the LaRCS `phase expression` which describes the entire parallel program in terms of its temporal computation and communication behavior. A `comtype` declaration describes a single communication edge; a `comphase` describes a set of synchronous communication edges. A `phase expression` instantiates all the edges of a parallel computation using the `comtype` and `comphase` declarations, and describes the message-passing behavior of the computation over time. We note that there is a distinction between “temporal communication behavior” (as used here) and dynamically evolving task graphs. In our model the entire graph is known statically. We describe temporal behavior by identifying collections of edges that are “active” simultaneously, and by the pattern of this activity.

Fig. 3 gives the LaRCS code for the *n-body* algorithm. The line numbers in the LaRCS code are used for reference purposes only in the following commentary:

1. **Name of algorithm and parameters.** The parameters specify the size of this instance of the parallel algorithm. The parameters for the *n-body algorithm* are `n`, the number of

```

# LaRCS code for the n-body algorithm

1.  nbody(n,s)
2.  attributes nodesymmetric;
3.  nodetype body
    labels 0..(n-1);
4.  comtype ring_edge(i) body(i) => body((i+1) mod n);
    volume = MSGSIZE;
    comtype chordal_edge(i) body(i) => body((i+(n+1)/2) mod n);
    volume = MSGSIZE;
5.  comphase ring
    forall i in 0..(n-1) {ring_edge(i)};
    comphase chordal
    forall i in 0..(n-1) {chordal_edge(i)};
6.  phase_expr
    {{ring |> compute}**(n-1)/2 |> chordal |> compute}**s;

```

Figure 3: LaRCS code for the n-body algorithm

bodies, and s , the number of iterations³.

2. **Attributes.** The programmer may specify global characteristics of the task graph, such as `nodesymmetric` or `planar`.
3. **Nodetype declaration.** A `nodetype` is defined by giving it a name, specifying the number of nodes, and specifying the node labeling. Node labels can be multi-dimensional and parameterized. For the *n-body algorithm* there is one `nodetype` declaration of type `body`. The nodes are labeled from 0 to $n-1$. If there is only one `nodetype`, the explicit declaration may be omitted.
4. **Comtype declaration.** A `comtype` specifies a single potential edge and can be parameterized. In both `comtype` and `comphase` declarations the symbol `=>` denotes unidirectional message passing and `<=>` denotes bidirectional message passing. In Fig. 3, there are two `comtype` declarations: `ring_edge` and `chordal_edge`. The `volume` field of the `comtype` declaration is an expression which specifies the message volume (typically in bytes) of a single message transfer.
5. **Comphase declaration.** A `comphase` identifies a potential set of edges involved in synchronous message passing, usually by specifying a set of values for the parameter(s) of one or more `comtypes`. The `comphase` declaration may itself be parameterized and these parameters are later instantiated within the phase expression. In the *n-body algorithm* there are two `comphase` declarations: `ring` and `chordal`.
6. **Phase expression.** The phase expression describes the temporal behavior of the computation in terms of its communication phases. Phase expressions are defined recursively below where r and s are phase expressions.
 - `compute` is a keyword phase expression denoting a computational phase of activity.
 - a single `comphase` is a phase expression.
 - sequence: $r \mid s$ is a phase expressions which denotes sequential execution of the phases.
 - sequential repetition: $r ** expr$ is a phase expression denoting repeated execution of r a number of times specified by arithmetic expression $expr$.

³Since LaRCS is intended for static mapping, we require an estimate of the run-time parameter, s .

- sequential loop: `for var = range { r }` is a phase expression denoting repeated execution of `r` a number of times specified by `range`, where `var` is a formal parameter in `r`.
- parallelism: `r || s` is a phase expression denoting parallel execution of phases `r` and `s`.
- parameterized parallelism: `forall var in range { r }` is a phase expression denoting parallel execution of phases in `r`, where `var` appears as a parameter in `r`.

Only expressions derived by a finite application of these rules are phase expressions. A precise definition of the semantics of phase expressions is given in [24].

3.1.1 The Benefits of LaRCS for Mapping

LaRCS plays a critical role in OREGAMI by (1) providing information about the regular structure of the parallel computation through the `comphase` declarations and the `phase expression`; and (2) by serving as an efficient representation of *families* of regular computation graphs whose size is independent of the size of the actual instantiated graph. The LaRCS compiler translates the LaRCS code into intermediate code (an abstract syntax tree) representation of the computation to be mapped. This intermediate representation can then be translated into the form needed for specific OREGAMI mapping algorithms such as the TCG, the static task graph for the whole computation, or the static task graph for a single `comphase`. OREGAMI provides a set of utility functions to generate specific data structures for these graphs such as an adjacency matrix, given the parameters that instantiate the size of the problem instance.

An example of the use of the LaRCS `phase expression` in mapping is described below. Further examples of the benefits of LaRCS for mapping are described in Section 4.

Most existing mapping algorithms, including several in OREGAMI, utilize the *static task graph* and require an estimate of communication volume (edge weights.) Current techniques for computation of edge weights include profiling, user estimates, and compiler analysis of the program code. In OREGAMI, the `phase expression` can be used to derive an arithmetic formula for calculation of communication volumes given the unit volume associated with a single message.

The LaRCS `comtype-declaration`'s `volume` field is typically initialized to the number of bytes sent in a single message of that `comtype`. For example, based on the code for the *n-body algorithm*, the volume for the `comtype ring` is 108 bytes. This value would be specified by the user in the LaRCS code. The `phase expression` can be used to compute the total communication volume for each communication edge in the static task graph by multiplying the unit volume times the number of iterations derived from the `phase expression`. Thus, the total communication overhead

for a ring edge of the *n*-body algorithm is equal to $108 \times n \times s$, where *n* and *s* are parameters instantiated at compile time.

At this stage in OREGAMI's development, the unit volume and all parameters must be expressed as integer constants; the phase expression then drives the calculation of the total communication volume for each comphase and for the complete static task graph. In the future, we will be expanding our system to accept volume declarations in terms of imported variables from the host language. This extension relieves the programmer of the burden of counting up the number of bytes in a message involving complex data structures and/or multiple data structures. In addition, we will examine the feasibility of mapping which utilizes *volume expressions* with uninstantiated variables.





3.2 MAPPER

In OREGAMI, mapping is usually achieved in three steps: *contraction* of the task graph to a smaller graph (in cases where the number of tasks exceeds the number of processors), *embedding*, assignment of the contracted clusters of tasks to processors, and *routing* of the messages through the interconnection network in order to minimize contention. Our mapping algorithms fall into three groups.

- **Class I:** Canned mapping algorithms for computations whose communication topology matches well-known graph families such as binary trees, binomial trees, rings, etc. Mappings for these computation structures for the hypercube, mesh, and deBruijn networks have been developed a priori using human ingenuity. Canned mappings are stored in a library and retrieved given the name and parameters of the computation graph and the network graph.
- **Class II:** Mapping algorithms for computations for which the regularity is expressed in LaRCS code. The mapping algorithms use techniques tailored for the specific regular properties captured by LaRCS.
- **Class III:** Mapping algorithms for arbitrary computations. Heuristic algorithms are used to compute the mapping in cases when no regularity can be exploited.

Figure 4 shows the organization of MAPPER and Table II summarizes the mapping algorithms we have developed, giving the general approach used, the performance results, and references to the relevant papers. These algorithms are describe in more detail in Section 4. OREGAMI is periodically updated with new mapping algorithms, including those developed by other researchers as well.

MAPPER ALGORITHMS

-  **contraction** grouping the tasks into clusters such that the number of clusters \leq number of processors
-  **embedding** assignment of clusters to processors
-  **routing** assignment of communication edges in the task graph to links in the network
-  **mapping** contraction + embedding + routing

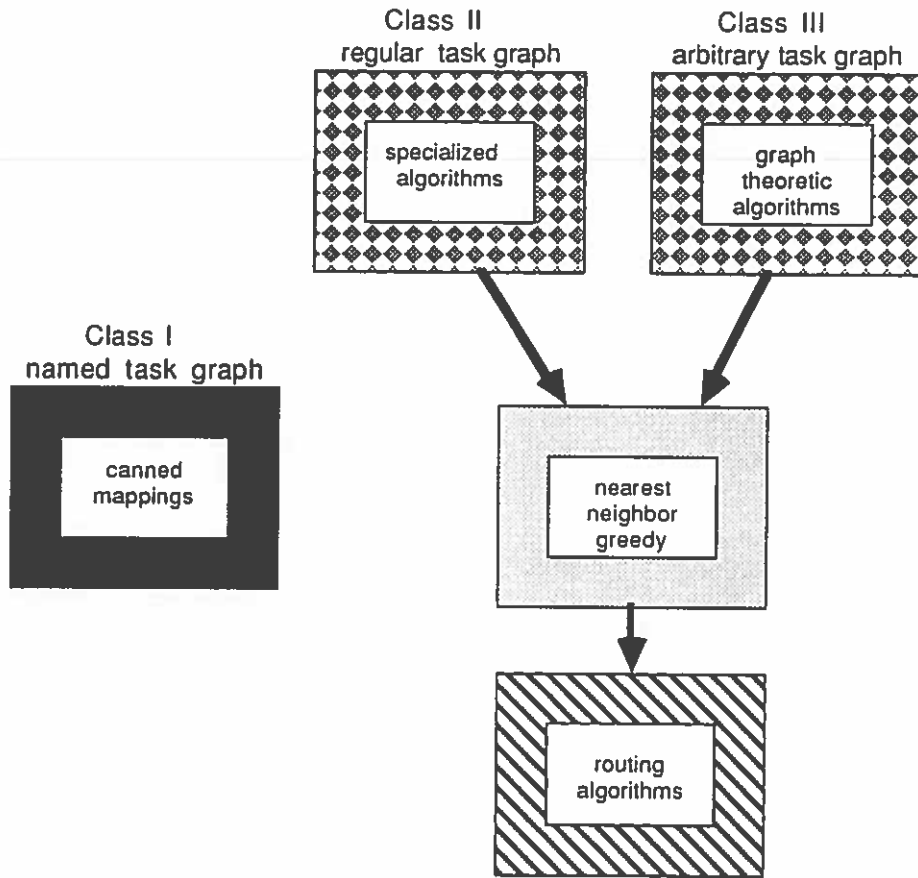


Figure 4: MAPPER contraction, embedding, routing

3.3 METRICS

The METRICS software allows the user to view and modify the mapping and routing produced by OREGAMI, while computing a wide range of performance metrics (see table below). METRICS is designed to display the mapping in a clear, logical, and intuitive format so that the user can evaluate it quantitatively (through well-known empirical performance metrics) as well as visually (through the use of colors and spatial layout, particularly when the computation graph exhibits regularity). METRICS is also designed to be interactive to allow the user to manually modify the mapping produced by the OREGAMI software. The target architectures currently supported by METRICS include the mesh and hypercube. METRICS supports analysis for three routing schemes: store-and-forward, virtual cut-through, and wormhole routing.

3.3.1 METRICS Displays

METRICS uses multiple windows to organize information into three selectable objects: the computation object, the architecture object, and the mapping object. Additionally, three subviews of the mapping object allow the user to focus on a specific processor, a specific link, or a single communication phase. Information from each of these objects and subviews is presented through three windows: text window, graphics window, and metrics window. (See Figure 5.)

The computation object corresponds to the parallel computation before mapping. The LaRCS program code is displayed in the text window. If the user selects the spatial perspective, the graphics window displays the computation's TCG as a static task graph. By default, the processes of the static task graph are laid out as a ring, but if a predefined computational structure, such as a tree, was specified by the LaRCS code, the graph will be displayed using the display semantics specific to that structure. The user may also view the computation as a DAG by selecting the temporal perspective. Communication phases are distinguished by color. The metrics window displays performance metrics such as the number of processes, the number of phases, the communication matrix with estimated message volume between processes (per phase and for the entire computation), and the execution matrix with estimated execution time of the processes.

The architecture object corresponds to the target machine. topology. The text window shows the name of the architecture, size parameters, routing scheme (store-and-forward, virtual cut-through, or wormhole), including values for channel bandwidth, message startup time, and message switching time. The graphics window displays a graphic representation of the bare The metrics window shows an adjacency matrix representation of the network topology, network metrics such as diameter and degree, and architecture characteristics such as channel bandwidth

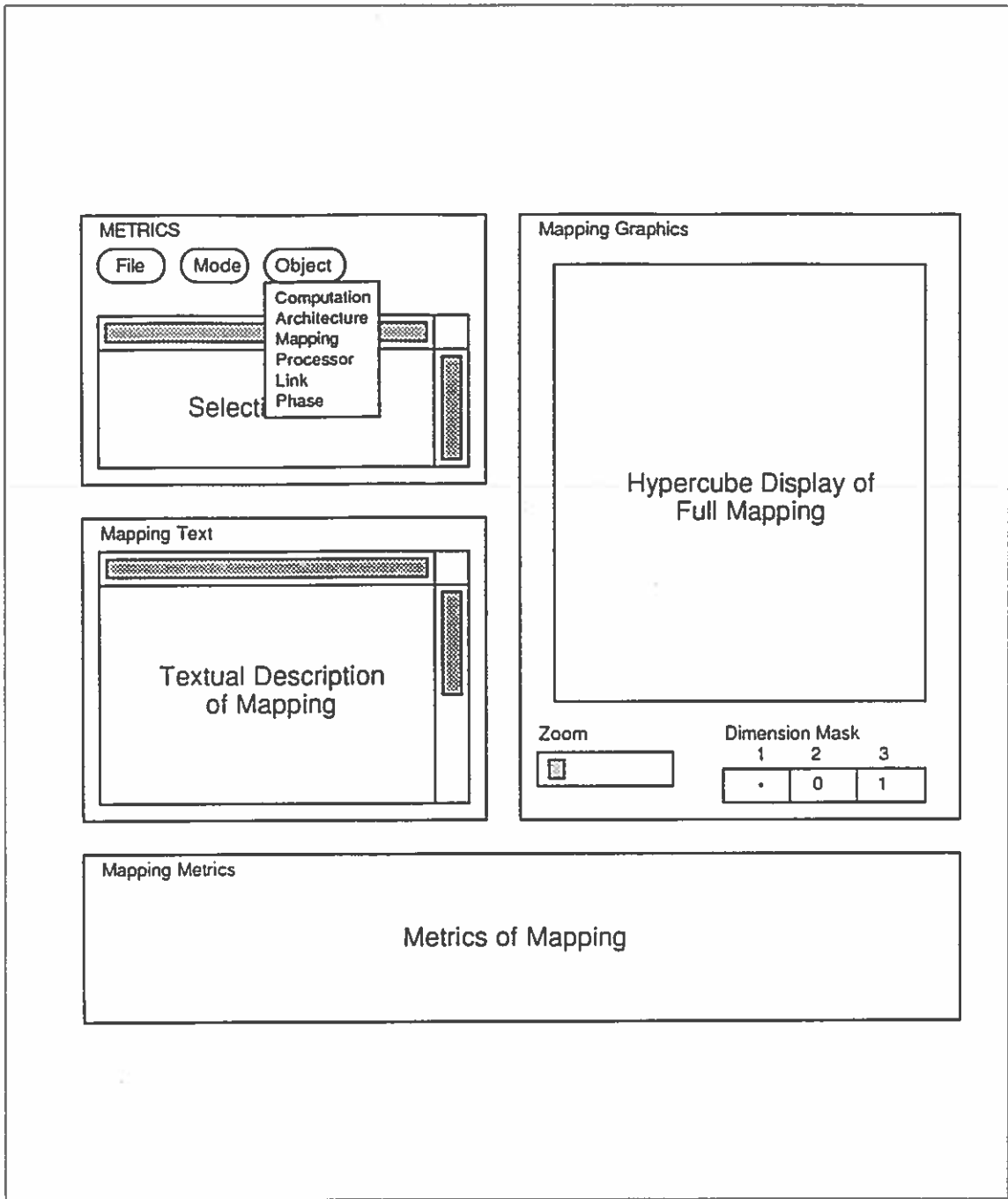


Figure 5: METRICS overview

and message-startup time.

The mapping object corresponds to the full computation as it is mapped onto the complete architecture. The text window identifies which processor each process is assigned to and enumerates each message routing as a list of processor elements. The graphics window displays the topology, with each process displayed in the appropriate processor element and messages rendered as edges, color-coded by phase. This view allows the user to rearrange the mapping by clicking on and dragging processes onto other processor elements. Selectable mapping metrics include processor task load, processor execution load, message dilation, link contention, and link communication volume, and latest finishing time (see Table I). Metrics are displayed as bar graphs, color-coded communication matrices, and color-coded topology views (see Figure 6).

The processor subview allows the user to examine a single processor and the set of processes mapped to it in detail. These windows show a graphic representation of the processor element, as well as the following metrics: task load, execution load, and comparisons to the global average.

The link subview focuses on a single architecture link connecting two adjacent processors, displaying the messages routed over that link. The link metrics include the total contention in messages and in communication volume, and a comparison to global averages.

The phase subview allows the user to examine a single communication phase in detail. The text window displays the LaRCS code for the communication phase, while the graphics window shows the topology and the mapped computation for the specified phase. Performance metrics for phases include those displayed for a complete mapping object.

The phase expression drives the calculation of completion time in a manner similar to the calculation of communication volume described above. Our current definition of completion time presumes barrier synchronization between phases *in order to simplify the calculation of this metric*. However, neither the architecture nor the parallel computation are constrained to execute with global synchronization between phases. The calculation of completion time of each phase on each processor takes into account the multiplexing of processes and the overhead of message passing including dilation, and contention. The computation of dilation and contention are based on the specific communication technology used for message-passing, taking into account message startup and switching times. METRICS currently models store-and-forward and wormhole routing.

| OREGAMI METRICS Performance Metrics | |
|-------------------------------------|---|
| Metric | Description |
| Processor Task Load | Number of tasks per processor non-zero-avg, avg, max, non-zero-min, min |
| Processor Execution Load | Sum of execution costs (node weights) per processor non-zero-avg, avg, max, non-zero-min, min |
| Dilation | Number of hops spanned by a communication edge non-zero-avg, avg, max, non-zero-min, min |
| Contention | Sum of communication costs (edge weights) per link non-zero-avg, avg, max, non-zero-min, min |
| Kandlur/Shin metric | Weighted sum combining dilation and contention $T(R) = \sum_{e \in E} (\sum_{m \in M, e \in R(m)} W(m))^2$ where, $T(R)$ is the cost of the routing function R , E is the set of links in the architecture, M is the set of messages, and $W(x)$ is the weight of a message x , $R(x)$ represents the set of edges on which R routes the message x . |
| Total IPC | Sum of communication costs (edge weights) for all inter-processor communication |
| Total Completion Time | $\sum_{p \in phases} \max \{c_p^1, c_p^2, \dots, c_p^m\}$ where $phases$ includes all instances of both compute phases and comphases, m is the number of processors, c_p^j is the completion time of phase p on processor j . |

Table I: METRICS performance metrics

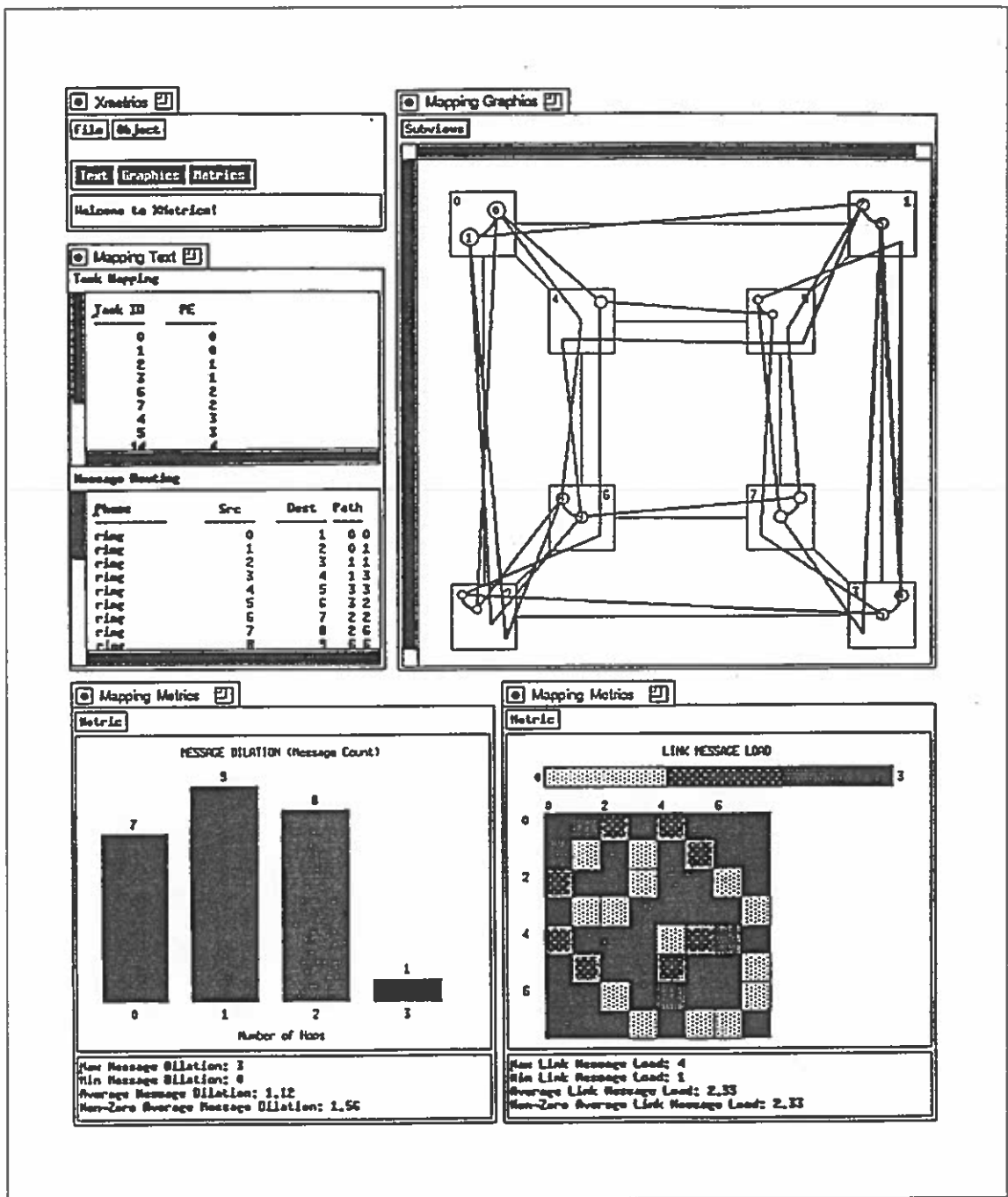


Figure 6: METRICS windows displaying the 15 body algorithm mapped to the 3-dim hypercube

4 OREGAMI Mapping Algorithms

In this section we describe the mapping algorithms we have developed for OREGAMI. These algorithms are summarized in Table II. Our algorithms perform contraction, embedding, and routing for both regular and arbitrary task graphs, utilizing a variety of mathematical techniques. The purpose of this discussion is to give the general flavor behind each of the algorithms and to show how information provided by LaRCS is utilized. A formal treatment of these algorithms and extensive performance evaluation is found in the referenced papers.

4.1 Canned mapping of binomial tree divide and conquer algorithms

Our contribution to the library of canned mappings is a set of mappings of the binomial tree [43] to the mesh and the deBruijn networks. As stated earlier, the input to OREGAMI for these canned mappings is simply the name of the computation graph and the architecture, plus parameters to instantiate the sizes of the graphs.

The binomial tree is a highly efficient structure for parallel divide and conquer algorithms, as an alternative to the full binary tree. Our mappings are illustrated in Figure 7. These mappings have constant time complexity because they are precomputed. They are proven to be optimal with respect to contention and we prove bounds on the average dilation (see Table II.) Space limitations prevents us from discussing the mappings in this paper; a formal treatment can be found in [25] and [49].

4.2 Contraction of Cayley Node Symmetric Task Graphs

We have developed a contraction algorithm for a subset of node symmetric task graphs called Cayley graphs which is made feasible by the information contained in the LaRCS comphase declarations. The algorithm is rooted in group theory [46] and yields a *symmetric contraction* in which there are an identical number of nodes per cluster and with each cluster having exactly one incoming and one outgoing 'contracted edge' for each communication phase. Each contracted edge represents an identical number of messages, thus the contraction is perfectly load balanced with respect to both computation and communication.

Our strategy requires that we detect whether the task graph T is a Cayley graph, defined as follows: For G a finite group and S a set of generators for G , $Cayley(G, S)$ is a graph where the nodes are the elements of G , and there is an edge with 'color' c from a to b if and only if there is a generator $c \in S$ such that under the group operation $ac = b$. No polynomial-time

| Class | Mapping | Technique | Complexity | Performance |
|-------|--|---|---------------------|--|
| I | binomial tree to hypercube [18] [25] | Gray code labeling | $O(1)$ | no contention avg. dilation ≤ 1 |
| I | binomial tree to mesh [25] | Gray code reflection | $O(1)$ | no contention avg. dilation ≤ 1.2 |
| I | binomial tree to deBruijn [49] | combinatorial/shift register sequences | $O(1)$ | no contention avg. dilation ≤ 2 |
| I | ring to deBruijn [35] | enumeration of necklaces/shift register sequences | $O(1)$ | gives precise no. necklaces |
| II | algorithms expressible as affine recurrences [32] (work in progress) | recurrences theory | $O(1)$ | optimal |
| II | Cayley/node symm. contraction [27] | group theory (Cayley graphs) | $O(VE)$ | optimal load balancing |
| III | arbitrary contraction [22] | maximum weight matching algorithm | $O(EV \log V)$ | minimizes IPC subject to load balancing; |
| III | store-and-forward routing [26] | maximal matching algorithm | $O(HDM^3)$ | minimizes contention |
| III | wormhole routing [47] | iteratively reduce contention; | $O(E M^2W^2d)$ | minimize contention; deadlock-free |
| III | wormhole routing [48] | shortest path algorithm | $O(M(nN + \log M))$ | minimize contention; deadlock-free |
| III | ecube routing | Gray code | $O(1)$ | deadlock-free |
| III | X-Y routing | x then y | $O(1)$ | deadlock-free |

Table II: OREGAMI mapping algorithms

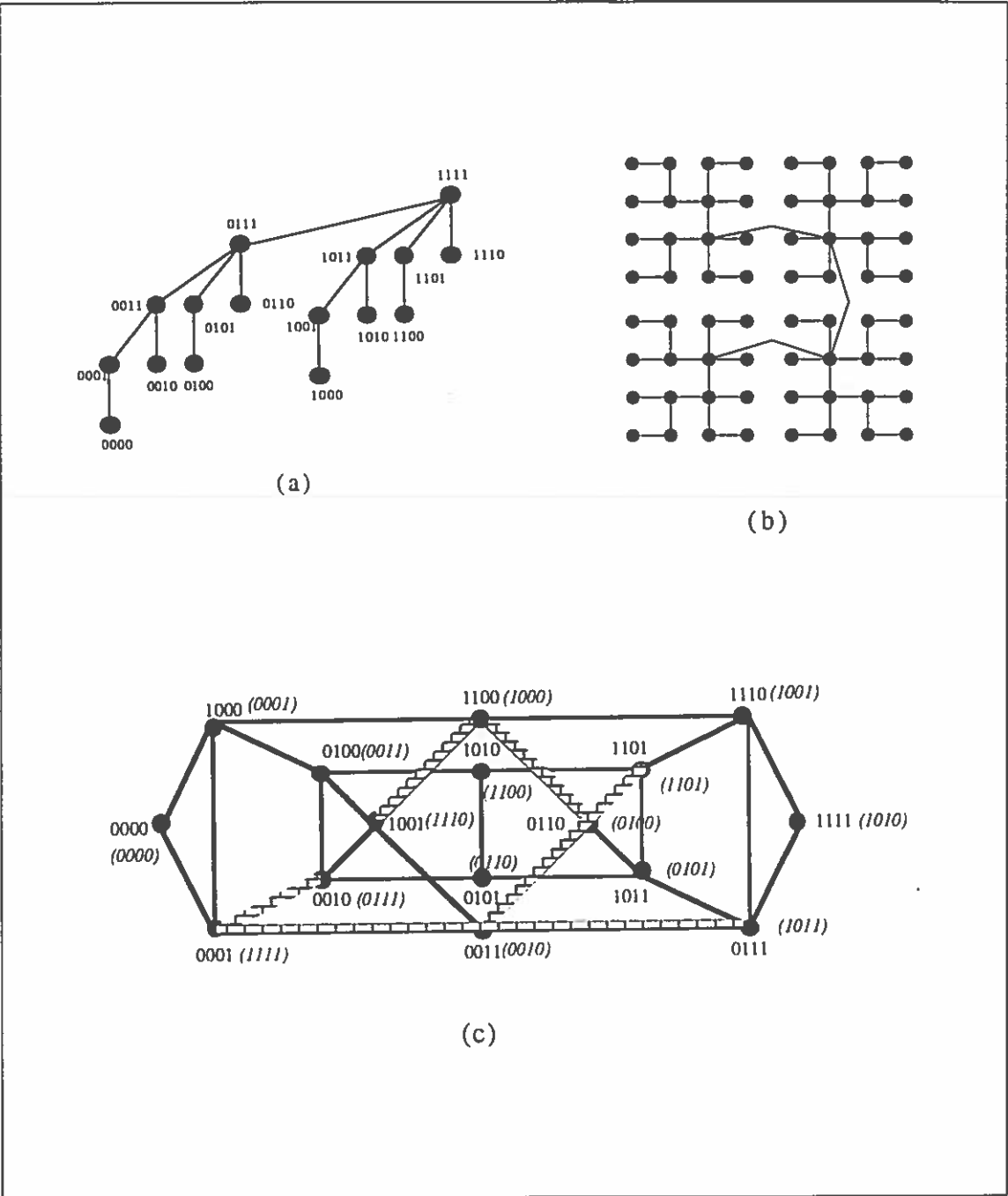


Figure 7: (a) Binomial tree and its canonical labeling (b) Mapping of 64 node binomial tree to 8×8 mesh (c) Mapping of 16 node binomial tree to 16 node deBruijn

algorithm is known that recognizes Cayley graphs based on an adjacency matrix representation [15]. However, with the aid of the LaRCS communication phases, for a task graph T with n nodes and m edges, in time $\mathcal{O}(nm)$, our algorithm either reports that T does not satisfy the criteria or produces a contraction as described above. The contraction algorithm is thus efficient only with the added information given by the LaRCS code. The algorithm is based on the fact that under certain conditions, the LaRCS communication phases of T can be used to directly derive the generators S of the underlying group G .

We note that this approach to contraction will be especially useful for data parallel algorithms which are inherently node symmetric. In addition, many interesting interconnection networks, such as the butterfly, hypercube, cube-connected cycles, are themselves based on Cayley graphs that have an underlying group structure [1]. Hopefully, this will also be an aid in the embedding and routing steps of the mapping.

Example: We will use the *8-node perfect broadcast algorithm* to illustrate the operation of Algorithm Cayley-Contract. (See Figure 8). In the *perfect broadcast algorithm*, 2^k processes with successive integer labels each disseminate information to all other processes in k steps by sending messages to neighbors whose distance from each sender represents successive powers of 2. In our example there are thus 3 communication phases, and process 0 will send to processes 1,2 and 4, in that order.

- The first requirement is that each communication phase is a bijection on the set of nodes $X = \{0, 1, ..7\}$. In that case, we calculate the value of each communication phase on the points of X and write the associated permutations in cycle notation as

```
comm1 = (0 1 2 3 4 5 6 7)
comm2 = (0 2 4 6)(1 3 5 7)
comm3 = (0 4)(1 5)(2 6)(3 7)
```

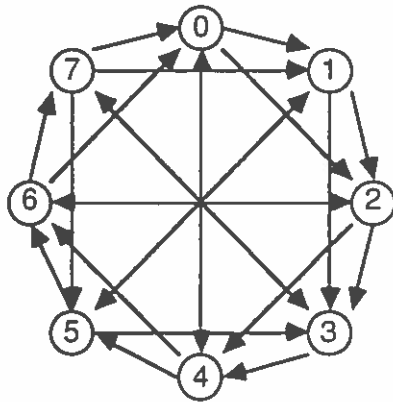
- The crucial point is that the communication phases can also be viewed as the set of generators of a permutation group G acting on X . This generator set gives rise to a unique Cayley graph CG . We can make use of the group G in the contraction of the task graph T when CG is isomorphic to T with the colors of CG and the communication phases in 1-1 correspondence. This is the case precisely when the action of G is *regular* on X . Detecting whether this is the case involves finding a spanning tree of T and thereby expressing elements of G in terms of products of generators. For our example the group

```

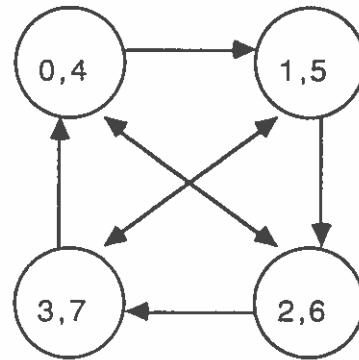
node_labels (0..7)
comm1 {task(i) => task((i+1)mod8)}
comm2 {task(i) => task((i+2)mod8)}
comm3 {task(i) => task((i+4)mod8)}

```

(a)



(b)



(c)

Figure 8: Group theoretic contraction: (a) Fragment of LaRCS code showing the communication types (b) Task graph (c) Contracted task graph

G is indeed regular and the correspondence between elements of the group generated and the nodes of the task graph is given as

| | | |
|------------------------------|----|--------|
| E0: (0)(1)(2)(3)(4)(5)(6)(7) | => | task 0 |
| E1: (0 1 2 3 4 5 6 7) | => | task 1 |
| E2: (0 2 4 6)(1 3 5 7) | => | task 2 |
| E3: (0 3 6 1 4 7 2 5) | => | task 3 |
| E4: (0 4)(1 5)(2 6)(3 7) | => | task 4 |
| E5: (0 5 2 7 4 1 6 3) | => | task 5 |
| E6: (0 6 4 2)(1 7 5 3) | => | task 6 |
| E7: (0 7 6 5 4 3 2 1) | => | task 7 |

- It can be shown that a quotient Cayley graph arising from a subgroup of G is a contraction of T preserving the symmetry of the parallel algorithm. We are interested in contractions where the number of clusters is close to the number of nodes in the graph of our target architecture, and where communication between tasks is internalized in a cluster. In our example, the target architecture has 4 processors and since the subgroup $\{E0, E4\}$ arises from the generator $comm3 = (04)(15)(26)(37)$ it yields a contraction where 2 messages are internalized in each cluster and we can map one cluster to each of the processors. Note that given this contraction, the lockstep symmetry in the execution of the algorithm is preserved by alternately multiplexing tasks in the order $\{0, 1, 2, 3\}$ followed by $\{4, 5, 6, 7\}$ on the four processors. This may be the most desirable property of these group theoretic contractions, which we hope to exploit further in the later stages of mapping and scheduling.

4.3 Contraction of Arbitrary Task Graphs

Algorithm MWM-Contract performs contraction of arbitrary task graphs based on the static task graph representation of the parallel computation which is obtained from the LaRCS code. MWM-Contract utilizes an $O(EV \log V)$ maximum weighted matching algorithm, where E is the number of edges and V the number of vertices in the static task graph. This contraction merges tasks into clusters such that the total interprocessor communication is minimized while satisfying the load balancing constraint that the total number of tasks per processor be bounded by some constant B . Typically B is set to the number of tasks divided by the number of processors.

If the number of tasks is less than or equal to twice the number of processors, the algorithm yields an optimal symmetric contraction. If the number of tasks is greater than twice the number of processors, a greedy heuristic is used in conjunction with the maximum weight matching

algorithm to find suboptimal task clusters. The greedy heuristic converts the original task graph into a smaller graph which satisfies the property that the number of new nodes is less than twice the number of processors. Then an optimal symmetric contraction can be found for this smaller graph, yielding a suboptimal contraction of the original task graph.

Preliminary testing of the performance of MWM-Contract involved simulations on a wide range of task graphs. The data used included four types of task graphs, with the best performance exhibited by graphs characterized by regular communication topologies (90.0% of the contractions found were optimal). Details of Algorithm MWM-Contraction, proofs of optimality, and simulation results can be found in [22].

Example: Figure 9 illustrates the operation of Algorithm MWM-Contract on an (irregular) computation in which 12 tasks must be assigned to 3 processors under the load balancing constraint of at most $B = 4$ tasks per processor.

- First, the greedy heuristic merges tasks into clusters until number of clusters is less than or equal to two times the number of processors. In order to satisfy the load balancing constraint of $B = 4$ tasks per processor, the greedy heuristic ensures that no cluster size exceeds $B/2 = 2$ tasks. This is achieved by examining edges in the task graph in non-increasing order based on the edge weights. Initially, each edge connects individual tasks. After several passes of the heuristic, however, an edge connects cluster of tasks which have been merged in previous passes. When an edge is examined, the two clusters are merged if the total number of tasks in the resulting combined cluster does not exceed $B/2$. For example in Figure 9, the edge with weight 15 does not result in merging because the combined cluster would have 4 tasks.

The outcome of the greedy heuristic is a new graph in which each node is a cluster of tasks from the original task graph. A single edge between two clusters will represent the total communication between all the tasks in the two clusters and will thus have a weight equal to the sum of the weights on the corresponding edges from the original task graph. In addition, the new graph will satisfy two conditions: (a) the number of nodes will be less than two times the number of processors and (b) the number of tasks within a cluster will be less than or equal to $B/2$.

- The maximum weight matching algorithm is then invoked on the new graph to produce an optimal contraction of clusters to processors which minimizes the total IPC and satisfies the load balancing constraint B . Figure 9(b) illustrates the contraction of the 6 clusters

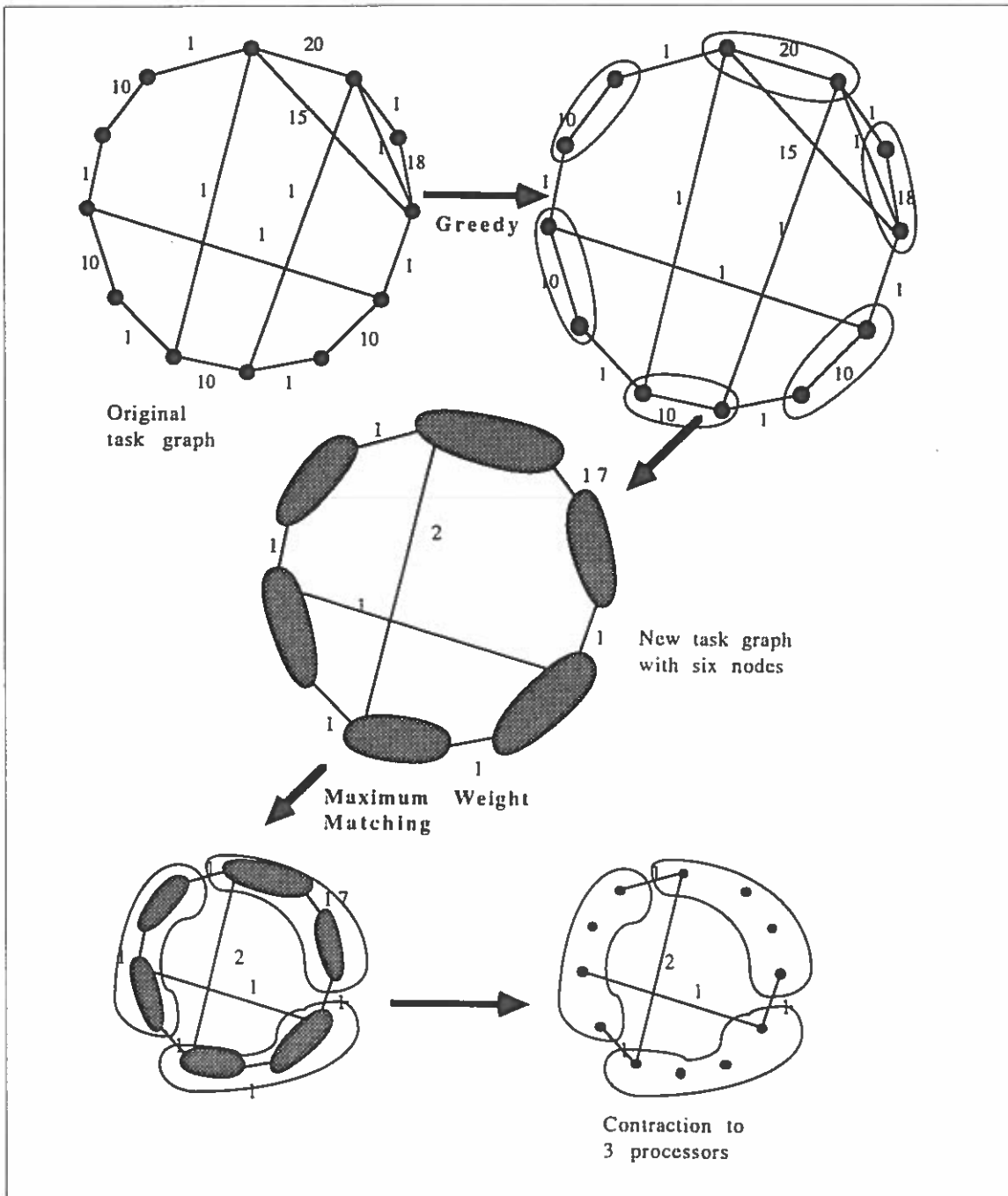


Figure 9: Algorithm MWM-Contract

to 3 processors and the corresponding contraction of the original 12 tasks. The total IPC = 6 and happens to be optimal in this case.

4.4 Embedding of Arbitrary Task Graphs

After contraction, embedding is achieved by Algorithm NN-Embed which uses a greedy approach to place highly communicating clusters on adjacent neighbors in the network graph.

NN-Embed is used by OREGAMI to assign process clusters to processors in the target architecture. Currently the mesh and the hypercube are supported. Its input is a contracted task graph which has no more clusters than the number of processors. The edge weights are assumed to represent communication volumes of a single dominant phase. We use a greedy heuristic that attempts to minimize the total weighted dilation of the embedding.

Given a contracted task graph, NN-Embed first constructs a list of all the edges in the graph sorted by weight. Ties are resolved by comparing the total weights on *all other* edges adjacent to the two edges. Then the algorithm traverses this list in linear time and for each edge, assigns its endpoints as follows:

- If both nodes have already been assigned, do nothing.
- If only one node has been assigned, then assign the other node to the closest free processor (this may not be unique, but the algorithm scans the processor list in a particular order—increasing integer labels for the hypercube and row major order for the mesh).
- If neither node has been assigned, randomly choose a free processor and assign one node to it and the other to its closest free neighbor.

Apart from the sorting step which took $O(V \log V)$ time, the rest of the algorithm runs linearly in the number of edges. In the future, we plan to develop embedding algorithms which exploit regularity.

4.5 Routing in OREGAMI

OREGAMI performs routing after the computation graph has been contracted and the tasks have been assigned to processors. Thus, the input to each of OREGAMI's routing algorithms specifies the set of messages to be routed by giving the source and destination processors corresponding to the sender and receiver processes, respectively. Our algorithms use the information provided by the LaRCS *comphase* declarations to achieve low contention routing. Routing directives are represented symbolically as a list of the source processor, intermediate processors, and

destination processors. Currently, OREGAMI does not translate these directives into system-specific routing control headers.

To date, we have developed three heuristic routing algorithms, one for systems which utilize store-and-forward routing `MM/SF-Route` and two for systems which utilize wormhole-like routing `WORM1-Route` and `WORM2-Route`. OREGAMI's routing library also includes the fixed `ECUBE` routing algorithm for the hypercube and the fixed `X-Y` routing algorithm for the mesh.

OREGAMI's routing algorithms perform routing on a phase by phase basis. Recall that the `comphase` declaration identifies logically synchronous communication, i.e. message passing that can occur simultaneously at runtime. This information enables the routing software to focus on only those messages that are capable of actual contention at runtime. There are several advantages to be gained by this approach:

- The likelihood of finding a routing with low contention is greater because fewer communication edges are considered.
- The use of contention as a performance metric for mapping and embedding is more accurate, since we avoid measuring *false contention*, i.e., when two edges are mapped to the same link which are not active simultaneously at runtime.

In this paper, we briefly describe Algorithm `MM/SF-Route` which uses an $O(HDM^3)$ bipartite matching algorithm to evenly distribute the edges of a single communication phase to the links of the interconnection network, where H is the diameter of the network, D is the max degree of a processor node, and M is the number of messages in the phase. `MM/SF-Route` is designed to minimize contention on a hop by hop basis and thus is suitable for architectures that use store-and-forward routing. However, we note that this algorithm can be used for systems which utilize wormhole routing schemes by including deadlock avoidance techniques such as virtual network partitioning. Algorithms `WORM1-Route` and `WORM2-Route` are described in [47] and [48].

Algorithm `MM/SF-Route` was tested for the hypercube by comparing its performance with random routing and the `ecube` routing algorithm [44]. The performance metric used in these experiments was *maximum* contention, i.e. maximum number of communication edges assigned to a single link. The experiments were performed on parallel computations from the OREGAMI test suite. Altogether 68 configurations were tested with the number of processors ranging from 8 to 32 and the number of tasks from 16 to 96. In 88.2% of the cases, the optimal routing was found. In 8.8% the routing was within 33% of optimal, and in the remaining 3% the routing was within 50% of the optimal value. We are currently performing additional experiments to test the performance of our routing algorithms using additional metrics for contention.

Example: Suppose the *15-body problem* is embedded on an 8-processor hypercube as shown in Figure 10. We discuss the operation of MM/SF-Route for the chordal edges only; the same procedure would be invoked for the ring edges also. In the bi-partite graph that is constructed by our routing algorithm, one partition consists of the communication edges in a single comphase and the other partition consists of the available (shortest) routes in the network that could potentially service these communication edges.

- In Figure 10(a), nodes in the task graph are labeled with the LaRCS task numbers, and links in the network are numbered arbitrarily from 1 to 12.
- In the chordal communication phase, task 0 sends to task 8, task 1 sends to task 9, etc. From a table of routing information for the 8-processor hypercube, we can determine the possible choices for the shortest routes: e.g., for messages from task 0 to task 8, possible routes are links 4 then 12, or links 9 then 8).
- From this information, we construct a bi-partite graph $G = (X, Y, E)$ where nodes in X represent chordal edges in the task graph, nodes in Y represent links in the hypercube, and edges in E connect each edge node to the links that can serve as the **first hop** (or only hop) in the possible routes from sender to receiver. Thus, in Figure 10(b), there is an edge from the node labeled (0-8) to the nodes labeled link 4 and link 9.
- A maximal matching of size M in graph G selects a maximal set of M distinct links in the hypercube and assigns M distinct chordal edges to those links. If $M \neq |X|$ then some nodes in X are unassigned. G is then reconfigured by removing all nodes of X covered by the matching and all edges incident to those nodes, and repeating the call to the maximal matching algorithm. Since each call to the maximal matching algorithm selects a given link at most once, we have achieved a low level of link contention.
- When all nodes in X are covered, a new graph is constructed for those chordal communication edges that have a choice of routes for the second hop. Note that in many cases, selection of the link for the first hop determines the second hop link.

5 Ongoing and Future Work

Areas for continuing work on the OREGAMI project include extensions to LaRCS to support dynamically spawned tasks and processor constraints on mapping; new and improved mapping algorithms, and scheduling directives. In addition, we plan to do additional performance testing

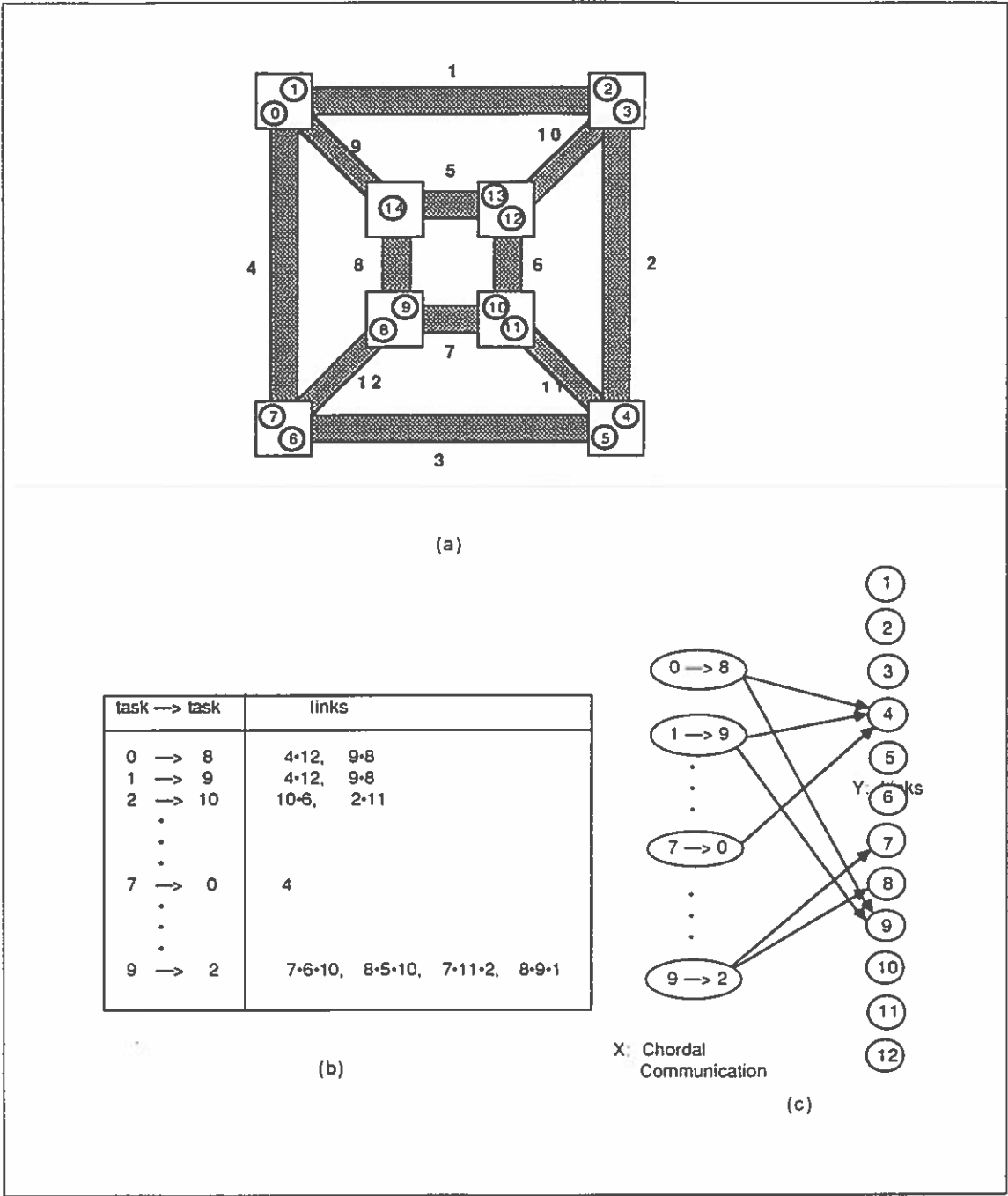


Figure 10: Algorithm MM/SF-Route: (a) 15-body algorithm mapped to 3 dim hypercube (b) table of possible shortest routes (c) bi-partite graph

both through simulation and empirical experiments using the new Sun/X-Windows OREGAMI tools.

LaRCS extensions: OREGAMI currently is designed only for computations in which the number of tasks is static. We plan to extend our software to handle computations with dynamically spawned tasks when the spawning pattern is regular and predictable. For example, parallel divide and conquer algorithms dynamically spawn tasks based on the size of the problem instance; however, it is known a priori that the spawning pattern will produce a full binary tree. We plan to augment LaRCS with the capacity to describe regular spawning patterns, and to design task assignment and routing algorithms to accommodate dynamically growing parallel computations. In addition, we will extend LaRCS to include language constructs that specify certain types of mapping constraints, such as the assignment of specific tasks to specialized processors (I/O processors, processors with floating point hardware, etc.).

Mapping algorithms: We will continue to augment the MAPPER library with new and improved algorithms for contraction, embedding, and routing. Some of the new approaches to mapping that we are investigating include algorithms for mapping computations expressible as affine recurrences; algorithms that perform two or more of the mapping steps simultaneously; algorithms that consider migrating processes at run time in order to accommodate phase shifts (as opposed to our current approach of finding one mapping that accommodates all the phases); and algorithms that avoid overspecification of communication topologies for common parallel paradigms such as aggregate and broadcast. For example, many parallel algorithms use a specific tree topology to aggregate results when a variety of alternate communication topologies will suffice (any spanning tree or the perfect broadcast ring of [17]). We would like to automatically select the aggregate topology that is 'compatible' with the topologies of other phases in the computation. Finally, we will continue to add to the library of 'canned' mappings for nameable task graphs.

Scheduling: Many parallel algorithms can be characterized as synchronous in nature, i.e., they are designed to run lockstep through their execution and communication phases. Therefore, it is advantageous to be able to coordinate the scheduling of tasks across processors after they have been assigned by MAPPER. We plan to extend OREGAMI to include a means for specifying *task synchrony sets* across processors. A task synchrony set is a set of tasks, one on each processor, that should be executing at the same time. This approach appears to be promising for computations whose task graph fulfills certain Cayley conditions as discussed in Section 4.2. Identification of these synchrony sets can be used to refine the routing algorithm and to produce local scheduling directives for each processor that ensure synchronous execution of the tasks in each set. The scheduling directives can be expressed in a notation similar to path expressions

[11] that specify the allowable ways to multiplex the tasks assigned to a given processor.

Testing: Our experiments thus far have focused on testing the performance of individual mapping algorithms. Our plans also include careful testing of the performance of the whole OREGAMI system by comparing OREGAMI mappings to random mapping, user manual mapping, and mapping produced by Berman's Prep-P system. These mappings will be evaluated using METRICS and through empirical experiments on the Intel iPSC/2.

Two key goals of our research are to provide tools that ensure portability of parallel software and to achieve the performance potential of parallel processing. OREGAMI was designed to achieve these goals by offering efficient algorithms for contraction, embedding, and routing, and by utilizing the user's knowledge through LaRCS and METRICS. The OREGAMI project is currently beginning its third year of development in which we hope to continue to contribute towards the development of an effective and practical tool for the mapping of parallel algorithms to parallel architectures.

References

- [1] S. B. Akers and B. Krishnamurthy. A group-theoretic model for symmetric interconnection networks. *IEEE Transactions on Computers*, C-38(4):555–566, April 1989.
- [2] D. A. Bailey and J. E. Cuny. Graph grammar based specification of interconnection structures for massively parallel computation. In *Proceedings of the Third International Workshop on Graph Grammars*, pages 73–85, 1987.
- [3] D. A. Bailey and J. E. Cuny. *Visual extensions to parallel programming languages*, pages 17–36. MIT Press, August 1989.
- [4] F. Berman. Experience with an automatic solution to the mapping problem. In *The Characteristics of Parallel Algorithms*, pages 307–334. The MIT Press, 1987.
- [5] F. Berman and L. Snyder. On mapping parallel algorithms into parallel architectures. *Journal of Parallel and Distributed Computing*, 4(5):439–458, October 1987.
- [6] F. Berman and B. Stramm. Prep-p: Evolution and overview. Technical Report CS89-158, Dept. of Computer Science, University of California at San Diego, 1989.
- [7] B.P. Bianchini and J.P. Shen. Interprocessor traffic scheduling algorithm for multiprocessor networks. *IEEE Transactions on Computers*, C-36(4):396–409, April 1987.
- [8] S. H. Bokhari. *Assignment Problems in Parallel and Distributed Computing*. Kluwer Academic Publishers, 1987.
- [9] J.C. Browne. Framework for fomulation and analysis of parallel computation structures. *Parallel Computing*, 3:1–9, 1986.
- [10] J.C. Browne. Code: A unified approach to parallel programming. *IEEE Software*, 6(4):10–19, July 1989.
- [11] R. H. Campbell and A. N. Habermann. *The Speification of Process Synchronization by Path Expressions*, volume 16, pages 89–102. Springer-Verlag, 1974.
- [12] Marina C. Chen. A design methodology for synthesizing parallel algorithms and architectures. *Journal of Parallel and Distributed Computing*, 3(6):461–491, December 1986.
- [13] M. H. Coffin. Par: An approach to architecture-independent parallel programming. Technical Report TR90-28, Dept. of Computer Science, University of Arizona, August 1990.

- [14] H. El-Rewini and T.G. Lewis. Scheduling parallel program tasks onto arbitrary target machines. *Journal of Parallel and Distributed Computing*, 9:138–153, 1990.
- [15] M. Fellows. Problem corner. *Contemporary Mathematics*, 89:187–188, 1989.
- [16] W. G. Griswold, G. A. Harrison, D. Notkin, and L. Snyder. Port ensembles: a communication abstraction for nonshared memory parallel programming. Technical report, Dept. of Computer Science, University of Washington, 1989.
- [17] Y. Han and R. Finkel. An optimal scheme for disseminating information. In *Proceedings of the 1988 International Conference on Parallel Processing*, pages 198–203, August 1988.
- [18] S. L. Johnsson. Communication in network architectures. In *VLSI and Parallel Computation*, page page 290. Morgan Kaufmann Publishers, Inc., 1990.
- [19] D.D. Kandlur and K.G. Shin. Traffic routing for multi-computer networks with virtual cut-through capability,. In *Preceedings of the 10th International Conference on Distributed Computer Systems*,, pages 398–405, May 1990.
- [20] Simon M. Kaplan and Gail E. Kaiser. Garp: Graph abstractions for concurrent programming. In H. Ganzinger, editor, *European Symposium on Programming*, volume 300 of *Lecture Notes in Computer Science*, pages 191–205, Heidelberg, March 1988. Springer-Verlag.
- [21] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.
- [22] V. M. Lo. Algorithms for static task assignment and symmetric contraction in distributed computing systems. In *Proceedings IEEE 1988 International Conference on Parallel Processing*, pages 239–244, August 1988.
- [23] V. M. Lo. Heuristic algorithms for task assignment in distributed systems. *IEEE Transactions on Computers*, 37(11):1384–1397, 1988.
- [24] V. M. Lo. Temporal communication graphs: Lamport’s process-time graphs augmented for the purpose of mapping and scheduling. Technical Report CIS-TR-92-05, University of Oregon, 1992. Submitted to *Journal of Parallel and Distributed Computing*.
- [25] V. M. Lo, S. Rajopadhye, S. Gupta, D. Keldsen, M. A. Mohamed, and J. Telle. Mapping divide-and-conquer algorithms ato parallel architectures. In *Proceedings IEEE 1990 International Conference on Parallel Processing*, pages III:128–135, August 1990. Also available as University of Oregon Technical Report CIS-TR-89-19.

- [26] V. M. Lo, S. Rajopadhye, S. Gupta, D. Keldsen, M. A. Mohamed, and J. Telle. OREGAMI: Software tools for mapping parallel algorithms to parallel architectures. In *Proceedings 1990 International Conference on Parallel Processing*, pages II:88–92, August 1990. Updated version available as University of Oregon Technical Report CIS-TR-89-18a and will appear in the *International Journal of Parallel Programming*.
- [27] V. M. Lo, S. Rajopadhye, M. A. Mohamed, S. Gupta, B. Nitzberg, J. A. Telle, and X. X. Zhong. LaRCS: A language for describing parallel computations for the purpose of mapping. Technical Report CIS-TR-90-16, University of Oregon Dept. of Computer Science, 1990. To appear in *IEEE Transactions on Parallel and Distributed Systems*.
- [28] J. Magee, J. Kramer, and M. Sloman. Constructing distributed systems in conic. *IEEE Transactions on Software Engineering*, SE-15(6):663–675, June 1989.
- [29] P. A. Nelson and L. Snyder. Programming paradigms for nonshared memory parallel computers. In *The Characteristics of Parallel Algorithms*, pages 3–20. The MIT Press, 1987.
- [30] C. D. Polychronopoulos. *Parallel Programming and Compilers*. Kluwer Academic Publishers, 1988.
- [31] C. D. Polychronopoulos. *Parallel Programming and Compilers*. Kluwer Academic Publishers, 1988.
- [32] Sanjay V. Rajopadhye and Richard M. Fujimoto. Synthesizing systolic arrays from recurrence equations. *Parallel Computing*, 14:163–189, June 1990.
- [33] A. L. Rosenberg. Graph embeddings 1988: Recent breakthroughs new directions. Technical Report 88-28, University of Massachusetts at Amherst, March 1988.
- [34] M. Rosing, R. B. Schnabel, and R.P. Weaver. The dino parallel programming language. Technical Report CU-CS-457-90, Dept. of Computer Science, University of Colorado at Boulder, April 1990.
- [35] R. Rowley and B. Bose. On necklaces in shuffle-exchange and de bruijn networks. In *Proceedings International Conference on Parallel Processing*, pages I:347–350, August 1990.
- [36] P. Sadayappan, F. Ercal, and J. Ramanujam. Clustering partitioning approaches to mapping parallel programs onto a hypercube. *Parallel Computing*, 13:1–16, 1990.
- [37] V. Sakar. Partitioning and scheduling parallel programs for execution on multiprocessors. Technical report, Ph.d. Thesis, Dept. of Computer Science, Stanford University, 1987.

- [38] C. L. Seitz. The cosmic cube. *Communications of the ACM*, 28(1):22–33, January 1985.
- [39] L. Snyder. Introduction to the configurable, highly parallel computer. *Computer*, 15(1):47–56, January 1982.
- [40] L. Snyder. *The XYZ abstraction levels of Poker-like languages*, pages 470–489. MIT Press, August 1989.
- [41] L. Snyder and D. Socha. Poker on the cosmic cube: the first retargetable parallel programming language and environment. In *Proceedings 1986 International Conference on Parallel Processing*, pages 628–635, August 1986.
- [42] H. S. Stone. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Transactions on Software Engineering*, SE-3(1):85–93, January 1977.
- [43] J. Vuillemin. A data structure for manipulating priority queues. *Communications of the ACM*, 21(4):309–315, April 1987.
- [44] C. L. Seitz W. J. Dally. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, 36(5):547–553, May 1987.
- [45] A. Wagner, S. Chanson, N. Goldstein, J. Jiang, H. Larsen, and H. Sreekantaswamy. Tips: Transputer-based interactive parallelizing system. Technical report, Dept. of Computer Science, University of British Columbia, 1990.
- [46] H. Wielandt. *Finite Permutation Groups*. Academic Press, 1964.
- [47] X. X. Zhong and V. M. Lo. Application specific deadlock free wormhole routing on multicomputers. Technical Report CIS-TR-92-03, University of Oregon, 1992. To appear in PARLE 92.
- [48] X. X. Zhong and V. M. Lo. An efficient heuristic for application specific routing on mesh connected multiprocessors. Technical Report CIS-TR-92-04, University of Oregon, 1992. To appear in 1992 International Conference on Parallel Processing.
- [49] X. X. Zhong, S. Rajopadhye, and V. M. Lo. Parallel implementation of divide-and-conquer algorithms on binary debruijn networks. Technical Report CIS-TR-91-21, University of Oregon, 1991. To appear in 6th International Parallel Processing Symposium.