OREGAMI: Software Tools for Mapping Parallel Computations to Parallel Architectures

Virginia M. Lo and Sanjay Rajopadhye
Samik Gupta
David Keldsen
Moataz A. Mohamed
Jan Telle

CIS-TR-89-18
January 19, 1990

Abstract

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. We have developed a unified set of software tools called OREGAMI for automatic and guided mapping of parallel computations to parallel architectures in order to achieve portability and maximal performance from parallel systems. Our tools include a description language which enables the programmer of parallel algorithms to specify information about the static and dynamic communication behavior of the computation to be mapped. This information is used by the mapping algorithms to assign tasks to processors and to route communication in the network topology. Two key features of our system are (a) the ability to take advantage of the regularity present in both the computation structure and the interconnection network and (b) the desire to balance the user's knowledge and intuition with the computational power of efficient combinatorial algorithms.

Department of Computer and Information Science
University of Oregon

OREGAMI: Software Tools for Mapping Parallel Computations to Parallel Architectures

Virginia M. Lo *and Sanjay Rajopadhye †
Samik Gupta
David Keldsen
Moataz A. Mohamed
Jan Telle

Dept. of Computer and Information Science
University of Oregon
Eugene, Oregon 97403-1202
503-686-4408
lo@cs.uoregon.edu

Sponsored by Oregon Advanced Computing Institute (OACIS) ‡

Keywords: mapping, routing, embedding, regular parallel computation, matching

Abstract

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. We have developed a unified set of software tools called OREGAMI for automatic and guided mapping of parallel computations to parallel architectures in order to achieve portability and maximal performance from parallel systems. Our tools include a description language which enables the programmer of parallel algorithms to specify information about the static and dynamic communication behavior of the computation to be mapped. This information is used by the mapping algorithms to assign tasks to processors and to route communication in the network topology. Two key features of our system are (a) the ability to take advantage of the regularity present in both the computation structure and the interconnection network and (b) the desire to balance the user's knowledge and intuition with the computational power of efficient combinatorial algorithms.

*Partially supported by NSF grant CCR-8808532
†Partially supported by NSF grant MIP-8802454
‡OACIS is a consortium of academic, industrial, and government agencies in the state of Oregon.

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 **maximal performance** from parallel software.

We have developed a unified set of software tools for automatic mapping of parallel computations to parallel architectures. Our tools include a description language which enables the programmer of parallel algorithms to specify information about the static and dynamic communication behavior of the computation to be mapped. This information is used by our mapping algorithms to assign tasks to processors and to route communication in the network topology. Two key features of our system are (a) the ability to take advantage of the regularity present in both the computation structure and the interconnection network and (b) the desire to balance the user's knowledge and intuition with the computational power of efficient combinatorial algorithms.

Fig 1 illustrates the components of the OREGAMI[1] system for mapping parallel computations to parallel architectures. OREGAMI is designed for use in conjunction with parallel programming languages that support a communication model, such as OCCAM, C*, and C and Fortran with communication extensions. The underlying architecture is assumed to consist of homogeneous processors connected by some regular network topology. Systems such as the iPSC/2, NCUBE, and INMOS Transputer are well-known candidates for use with OREGAMI. Our software tools prototype is implemented in MacScheme and Think-C on a MacII using color displays for visualization of the mapping.

OREGAMI has three main components:

LaRCS (**La**nguage for **R**egular **C**ommunication **S**tructures) is a language that allows the user to express (a) the structure of the parallel computation graph using simple functions, and (b) the dynamic 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 many parallel programming languages to provide information about regularity to be found in the communication structure of the computation. LaRCS may be viewed as a notation for *statically* describing the communication behavior of a computation. The information in a LaRCS specification is used by the MAPPER and METRICS software.

MAPPER, our mapping software is a library of graph theoretic and group theoretic algorithms which map the computation to the architecture. The specific algorithm that is invoked depends on the information provided by the LaRCS description, and falls into three classes. Many parallel algorithms have well known communication structures (such as trees,

[1] For University of **OREG**on's contribution to the elegant symmetric structures (contractions) produced by ori**GAMI** paper folding.

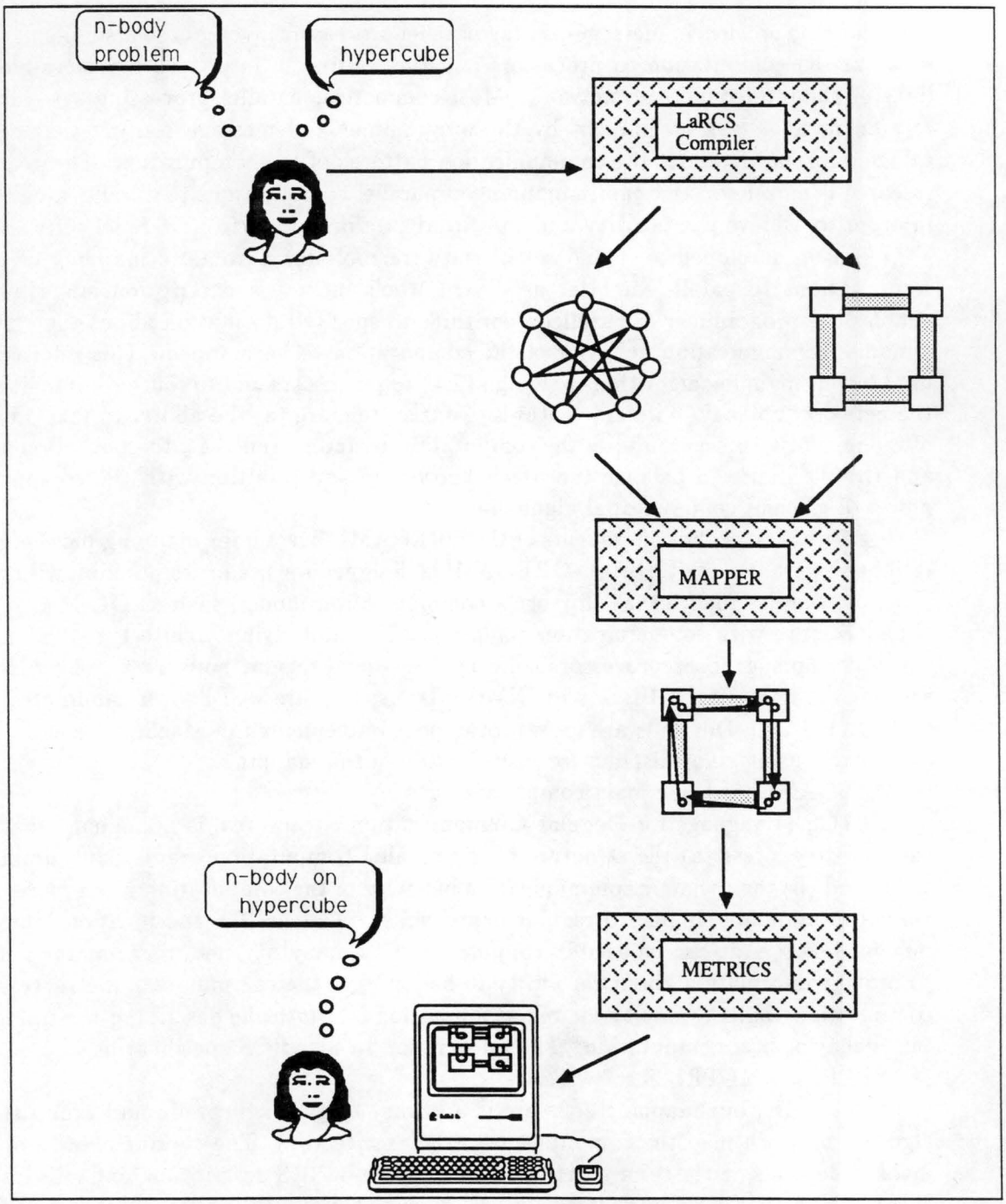


Figure 1: Overview of the OREGAMI System

hypercubes, 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 that corresponding to permutation groups), or are intended to be mapped to specific class of architectures. For such computations, OREGAMI uses specialized algorithms which are often extremely efficient (for the group theoretic mappings, this simply involves determining the factors of the order of the group). If no such regularity can be exploited, we perform the mapping in three steps – contraction, embedding, and routing, using efficient polynomial time heuristics.

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 allows the user to focus on specific processors or links and provides the opportunity for manual modification of the mapping.

The specific contributions of our work include

- A new graph model for parallel computations that expresses both static and dynamic behavior of the computation. Our model integrates the static task graph model used for static task assignment and mapping [Sto77],[Bok87], [BS87] with the precedence constrained DAG model used in multiprocessor scheduling research [Cof76], [Pol88].
- The LaRCS description language and compiler which allows the user to specify both static and dynamic information about the parallel computation in a natural, high-level, parametric notation, enabling the mapping algorithms to make optimal and near optimal task assignment and routing decisions.
- An innovative approach to graph contraction using techniques from group theory. The group theoretic contraction algorithm can be used on node symmetric task graphs and runs in polynomial time.
- Algorithms for contraction, embedding and routing using maximum weight matching and maximal matching algorithms. These algorithms produce optimal results under conditions exhibited by a wide range of parallel algorithms and run in polynomial time.

This paper provides an overview of the OREGAMI mapping tools. Section 2 presents our graph model for parallel computations, basic definitions, and describes related research. Sections 3, 4, and 5 describe the three components of our system: the LaRCS description language and compiler, the MAPPER, with its contraction, embedding, and routing algorithms, and METRICS, the interactive user interface for display, analysis, and user modification of mappings. Section 6 discusses areas of on-going and future work on this project.

2 Definitions and Related Research

We consider parallel computations which can be characterized as a static set of communicating tasks, i.e., we do not currently consider computations that dynamically spawn subtasks[2]. Our graph model is designed to capture two types of regularity exhibited by parallel computations: (a) regularity in the communication topology, i.e., which tasks communicate through message passing and (b) regularity in the execution and communication patterns over time which we call the algorithm's *phase behavior.*

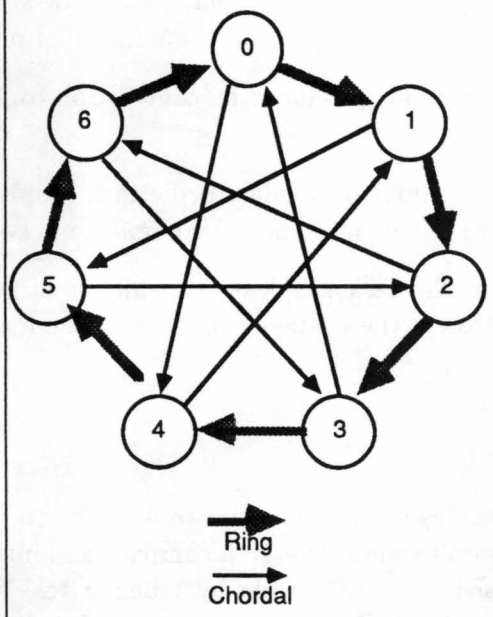
More precisely, we model the parallel computations as a weighted and colored directed graph $G = (V, E_1, E_2, ..., E_c)$ in which each task t_i is represented by a node $v_i \epsilon V$. Each edge set $E_k, k = 1, ..., c$ corresponds to one communication phase of the computation and is conceptually assigned a unique color (actual, physical colors are used by METRICS to display the phase behavior). Each directed edge $e_{ij} \epsilon E_k$ represents the fact that messages are sent from task t_i to task t_j during phase k of the computation. Weights on tasks nodes represent the (approximate) execution time of the corresponding task, and weights on the edges represent the message volume transmitted during the corresponding communication phase. These weights may be functions of the incident nodes and other parameters of the problem. However, if the graph is regular, its LaRCS description is very compact — an order of magnitude smaller than the size of the graph. We have found that a large number of practical problems exhibit such regularity. Moreover, for many such graphs there exist efficient mapping methods which exploit the regularity by reasoning with this compact notation rather than the entire graph. We present two such classes to support our claim.

Our model of computation is also well-suited to the *data parallel* model of parallel computation(also called the *Single Code Multiple Data,* or SCMD model of parallelism). This school holds the view that the best way to program multicomputers is to distribute the data over the memories of the processors and have each processor execute the same program [HS86,Joh87]. The OREGAMI system may be viewed as an attempt to map a data parallel program to a parallel architecture, using the compact LaRCS description to obtain the mapping *efficiently.*

Fig 2a illustrates our task graph model for a parallel algorithm for the *n-body problem* which was developed for the Caltech Cosmic Cube [Sei85]. We will use this example throughout this paper to illustrate the OREGAMI system.

> The *n-body problem* requires determining the equilibrium of n bodies in space 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 neigh-

[2]However, we do handle special instances of dynamic tasks, where the spawning pattern is predictable at compile time.



(a)

nbody(n)
imports(s, *compute*1, *compute*2, *msgsize*);
nodetypes *body* ;
nodeattr *nodesym*;
numnodes n ;
comtypes *ring*, *chordal* ;
exectypes *compute*1, *compute*2 ;
body{
numnodes n;
labels − **base** 10 ;
nodelabels (0..($n-1$));
comtypes *ring*, *chordal* ;
}
ring {
body(i) ⇒ *body*(($i+1$)*mod* n);
volume = *msgsize*;
}
chordal {
body(i) ⇒ *body*($i+(n+1)/2$)*mod* n);
volume = *msgsize*;
}
phase − **exp** : $[(ring.compute1)^{\frac{n-1}{2}}.chordal.compute2]^S$
end

(b)

Figure 2: The n-body problem: (a) The Task Graph, (b) LaRCS Description

bor 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.

The work presented here is also related to that of Berman and Snyder [BS87,Ber87] who also exploit regularity of communication behavior (by expressing it in terms of a context free grammar called an *edge grammar*). Bailey and Cuny [BC86] and Kaplan and Kaiser [KK] have used *graph grammars* to describe regular communication topology for purposes of mapping or in order to preserve structured code. LaRCS provides an alternative to these grammar-based approaches which we feel is easier for the programmer to use. The 'canned' mappings described in Section 4 include graph embedding results from [FF82], [Ros88], [GH88]. For one class of regular computations, the mappings we present are similar to those used in systolic array synthesis [CS84,Che86,RF88]. For arbitrary graphs, our mapping algorithms are similar to those of Stone [Sto77] and Bokhari [Bok87] because of their foundation in network flow algorithms.

For the remainder of the paper, we will use the following terminology:

- **Contraction** is the partitioning of the vertices of the computation task graph into clusters such that the number of clusters is no more than the number of processors.
- **Embedding** is the assignment of the clusters produced by contraction to processors, with at most one cluster per processor.
- **Routing** is the assignment of each communication edge in the task graph to a path in the network. The route may involve one or more hops through the network.
- **Dilation** of a communication edge in the task graph is the number of hops in the routing of that edge. **Average dilation** is the average over all communication edges in the task graph.

3 LaRCS

The LaRCS graph description language enables the programmer to describe the static and dynamic communication structure of the parallel algorithm in a compact and intuitive way. Static communication topology is expressed in LaRCS code by labeling task nodes and by representing communication edges as a simple function of the node labels. Dynamic communication behavior is represented by a *phase expression* which is similar to regular expressions. LaRCS code is much more space-efficient than an adjacency matrix since it allows parametric descriptions (i.e., size of the description is independent of the number of nodes in the task graph). LaRCS allows the user to conveniently express information about the regularity [3] of the task graph for use by MAPPER's algorithms, thus relieving the mapper of the burden of detecting meaningful regular patterns (a computation which is NP-hard in some cases). As discussed earlier, the LaRCS description is language-independent and can be used with a wide range of parallel programming languages by importing a few variables from the host language.

Below, we illustrate the use of LaRCS to describe the *n-body problem*. The numbers below correspond to the numbered items in Fig 2b. A full discussion of LaRCS appears in [LRG+].

1. name of algorithm and parameters. The only parameter for the *n-body problem* is n, the number of bodies.
2. imported variables from the algorithm source code
3. node declaration

   For the *n-body problem* there is one `nodetype` declaration of type *body*. The nodes are labeled from 0 to $n-1$ in decimal and the graph is `nodesymmetric`. Other LaRCS labeling schemes include binary, multi-dimensional numbering (e.g. for arrays of processes), and parameterized numbering schemes.

[3] Our definition of *regularity* includes intuitive notions such as symmetry, parameterizability, recursive structures, and uniformity.

4. communication phase declarations

   A communication phase in LaRCS identifies a set of edges involved in synchronous message passing in the parallel computation. The communication phase declaration describes the edges in one phase of the task graph using intuitive functions. These functions may involve arithmetic expressions, for-loops, while-loops, imported parameters, and other LaRCS variables.

   The *n-body problem* has two communication phases: *ring* communication and *chordal* communication. Both communication types use the same node labeling scheme, and simple arithmetic expressions suffice to describe the ring and chordal edges:

   ring: $i \rightarrow (i+1) \bmod n$
   chordal: $i \rightarrow (i + \frac{n+1}{2}) \bmod n$

   In general, it may be necessary to use multiple labeling schemes, one for each communication type. The user must then give the correspondence between the node labeling schemes. The communication volume is the amount of data sent in one message on the associated edge; it can be a parameterized arithmetic expression, or actual amount in bytes, or *sizeof* imported data structures.

5. Execution phase and execution costs

   An execution phase corresponds to a body of code bracketed by two successive communication phases. Execution costs for each execution phase are estimated by the user, the compiler, or derived from runtime monitoring of the computation.

6. phase expression

   The phase expression describes the dynamic behavior of the computation in terms of its execution and communication phases. Phase expressions are a generalization of a simple notation that has been used to describe the compute-aggregate-broadcast paradigm [NS87]. For the *n-body problem*, the phase expression is given by:

   $((ring; compute1) \wedge \frac{n+1}{2})chordal; compute2) \wedge s$

   Phase expressions are defined recursively below where r and s are phase expressions.

   - ϵ is a phase expression denoting an idle task.
   - a single communication phase or execution phase is a phase expression.
   - sequence: $r;s$ is a phase expression which denotes sequential execution of the phases.
   - repetition: $r \wedge expr$ is a phase expression denoting repeated execution of r a number of times specified by expression *expr*. *expr* can be an arithmetic expression, a logical expression, or a parameterized *for loop*.

- parallelism: $r \parallel s$ is a phase expression denoting parallel execution of phases r and s.

The LaRCS compiler translates LaRCS code into a set of Scheme functions that are used by a generic library of functions called by MAPPER and METRICS. These generic functions are used to build the data structures needed for the mapping algorithms and for computing the mapping performance metrics. Some of the LaRCS output for the n-body problem is shown in Fig 2c. LaRCS has been used to describe a wide variety of parallel algorithms including matrix multiplication, fast Fourier transform, topological sort, divide and conquer using binomial trees, simulated annealing, Jacobi iterative method for solving Laplace equations on a rectangle, successive over-relaxation iterative method, and perfect broadcast distributed voting.

4 MAPPER

MAPPER consists of a library of algorithms to perform contraction, embedding, and routing of the task graph. The current configuration of MAPPER is shown in Fig 3 and explained below. As our research continues, we plan to replace and augment the algorithms in the MAPPER library. MAPPER handles three classes of task graphs:

4.1 Nameable Task Graphs

These graphs can be described as belonging to a well-known graph family such as ring, mesh, hypercube, full binary tree, etc. In this case, contraction and embedding can often be accomplished in constant time by hashing on the name of the task graph and the name of the network topology to lookup a precomputed mapping. A number of researchers have contributed to the set of task graph/topology pairs for which "canned" contraction, embedding, and/or layout can be achieved (see Fig 3). Our contribution to this group is an embedding of the binomial tree to the square mesh. In [LRG+89] we show that the binomial tree is ideally suited to the general class of parallel divide and conquer algorithms and show an embedding that has average dilation bounded by 1.2 for arbitrarily large binomial tree and mesh. We conjecture that this mapping is optimal with respect to average dilation.

4.2 Computation Graphs with Regular Structure

As mentioned earlier, many algorithms often have a very regular structure, and OREGAMI is intended to exploit this regularity. Regularity implies a computation whose communication structure can be *parameterized* in such a manner that a *finite* LaRCS program can describe arbitrarily large communication graphs. If a computation is regular, we can often determine efficient methods to optimally map it to certain target architectures. Such methods are efficient precisely because they the (compact) LaRCS program (rather thatn the entire communication graph, which is an order of magnitude larger). Often these methods are $O(1)$ (i.e., independent of the size of the computation graph) or $O(n)$. We will

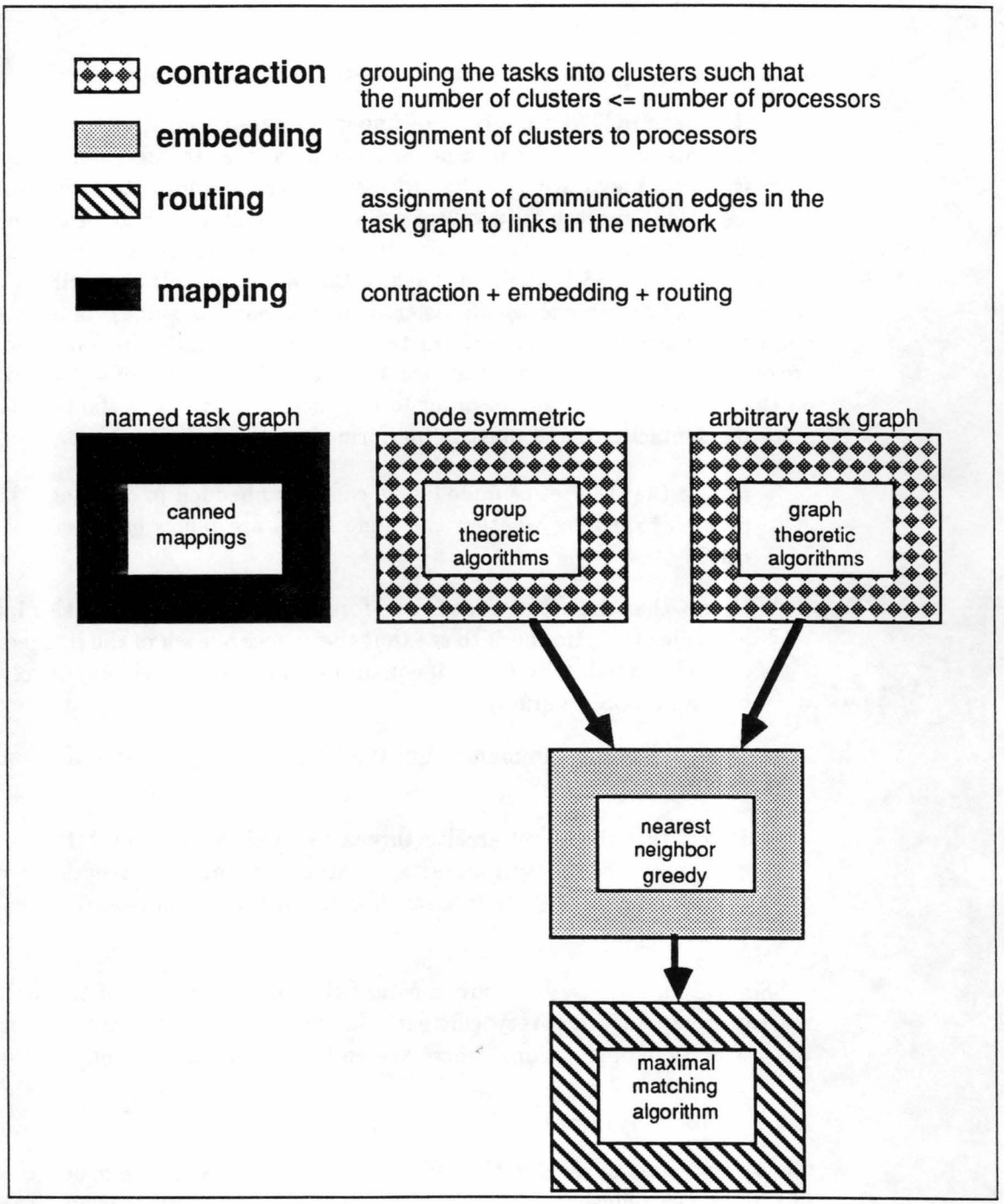


Figure 3: MAPPER Algorithms

now illustrate two classes of regular computations for which we have obtained such efficient mapping methods.

4.2.1 Mapping Affine Recurrences to Systolic Arrays

Systolic Arrays [KL80] are a class of fine-grain parallel architectures which consist of very simple, identical processors in a nearest-neighbor, planar interconnection (such as a mesh, a hexagon or a linear array). The processors operate in lock-step synchrony (although this is not essential) and the data values circulate through the array in a pipelined fashion.

Recently there have been advances in "compiling" high-level algorithmic specifications to systolic arrays [RF88,FM85]. In much of this work, the initial specifications are expressed as *systems of recurrence equations* that have linear (or affine) data dependencies, and a number of algorithms that perform the mapping efficiently are available. These methods are efficient precisely because they treat the data dependency of the algorithm as a *function* on the nodes of the graph. In order to use these algorithms in the OREGAMI system, the following syntactic checks must be performed on the LaRCS program.

- Detect that the set of node labels can be embedded in an integer lattice space. This reduces to testing whether the node labels are *tuples of integers*, a simple syntactic check.
- Detect that the set of node labels form a convex polytope in this lattice. This is also a simple syntactic check to see that the *ranges* of each of the integers used in the label sets are bounded by linear inequalities (the inequalities may involve the parameters of the LaRCS program).
- Detect that the communication types in the body of the LaRCS program are affine functions.
- Detect that the target architecture ia a systolic array or an MIMD mesh. In the latter case, many of the systolic array synthesis algorithms, together with the results on partitioning large systolic arrays for smaller sized hardware) can be used to perform the mappings.

Since each of these tests are constant time compiler tests of the LaRCS program, the resulting mappings are very efficient. In fact many of the methods are used for signal processing type applications, where the entire computation graph may be infinite!

4.2.2 Node Symmetric Task Graphs

The second class of regular MAPPER algorithms are rooted in group theory and make use of Cayley's graphical representation of a group. The algorithm is applicable to task graphs, T, that are Cayley graphs, a subset of the node symmetric graphs. It is based on the fact that given certain conditions, the LaRCS communication functions T can be used to directly derive the generators of the underlying group G. Each quotient group of G will give

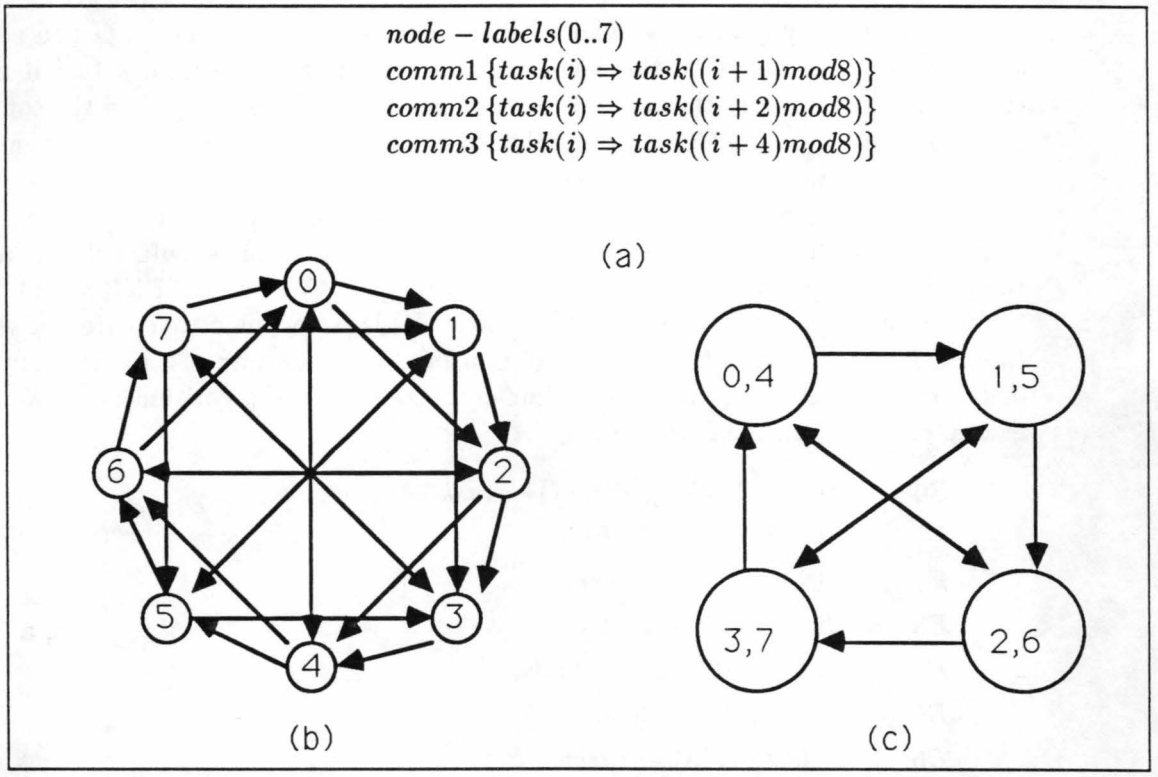


Figure 4: Group Theoretic Contraction: (a) Fragment of LaRCS code showing the communication types, (b) its Task Graph, (c) Contracted Task Graph

rise to a Cayley graph that is a contraction of T with an identical number of nodes in each cluster, and with an identical number of messages of each communication type mapped to each edge so that embedding and routing is simplified. Hence the mapping problem reduces to determinig the subgroups of G. To illustrate the ideas, we will use the task graph of the *8-node perfect broadcast algorithm*, where 8 tasks must "elect a leader," as shown in Fig 4a.

The first requirement is that each communication function is a bijection on the set of nodes $X = 0,..7$. In that case, we calculate the value of the communication function on each member of X and write the associated permutations in cycle notation as follows.

$$\begin{aligned} comm1 &= (01234567) \\ comm2 &= (0246)(1357) \\ comm3 &= (04)(15)(26)(37) \end{aligned}$$

meaning that $comm2(0) = 2, comm2(2) = 4, comm2(4) = 6, comm2(6) = 0$ and so on. The crucial point is that *comm1*, *comm2* and *comm3* can also be viewed as the set of generators

of a permutation group G that acts on X[4]. The set of generators of the group G gives rise to a unique Cayley graph CG, whose nodes correspond to elements of G and edges to the action of the generators on the elements, i.e. there is an edge from a to b if and only if there is a generator c so that $ac = b$. We can make use of the group G in the contraction of the task graph T when CG is isomorphic to T. This is the case precisely when the action of G is regular on X. We then have $|G| = |X|$ and G is transitive on X, so G and X are in one-to-one correspondence, defined for each $g \in G, g \leftrightarrow g(x)$ for any fixed $x \in X$. By convention, we let x be the smallest member of X. Since this should hold for any $x \in X$, the cycles of g, $g \in G$, should all be of equal length. Conversely, if $|G| = |X|$ and all the elements of G have equal-length cycles, then CG is isomorphic to T. Hence, we compute the cycle notation of all the elements of G using the generators. This is the dominant part of the computation, and hence the time complexity of the algorithm is $O(|X|^2)$. For our example, we obtain the following.

$$
\begin{aligned}
E0 &= (0)(1)(2)(3)(4)(5)(6)(7) \leftrightarrow task0 \\
E1 &= (01234567) \leftrightarrow task1 \\
E2 &= (0246)(1357) \leftrightarrow task2 \\
E3 &= (03614725) \leftrightarrow task3 \\
E4 &= (04)(15)(26)(37) \leftrightarrow task4 \\
E5 &= (05274163) \leftrightarrow task5 \\
E6 &= (0642)(1753) \leftrightarrow task6 \\
E7 &= (07654321) \leftrightarrow task7
\end{aligned}
$$

We see that $|G| = 8 = |X|$ and all elements have equal-length cycles. The interested reader can check that T is isomorphic to CG by applying the generators to the elements and drawing the Cayley graph. The point to note now is that any normal subgroup H of G will yield a quotient group G/H where an element of G/H corresponds to a coset of G. Moreover, each coset is of the same size, and so the Cayley graph of G/H will be a contraction of T with an equal number of tasks in each cluster. We are primarily interested in contractions where the number of clusters is close to the number of nodes in the graph of our target architecture A, and where some communication between tasks is internalized in a cluster (this yields a uniform load balance). A corollary to Sylow's theorem tells us that if $|T|/|A|$ is the power of a prime, then a contraction exists where the number of cosets (or clusters) is exactly $|A|$. A quotient group arising directly from a generator with cycle-length l, will yield a contraction that internalizes l messages (or edges). In light of this, if we say A is a hypercube of 4 processors, for our example we have $|T|/|A| = 8/4 = 2^1$ and 2 is a prime, so a perfectly balanced contraction exists, and the subgroup $\{E0, E4\}$ from the generator $comm3 = (04)(15)(26)(37)$ yields a contraction where 2 messages are internalized in each cluster. This is the clustering shown in Fig 4c.

[4]By convention, we adopt left-to-right composition of functions, so that (123) composed with (13)(2) gives (12)(3).

Note that the input LaRCS program may not have a communication graph that is a Cayley graph and we will therefore have to detect whether the algorithm outlined above can be used to obtain a valid contraction. This can be done if we observe that, in general, G can have up to $|X|!$ elements, but for our purposes we can halt the computation as soon as the number of elements in any cycle exceeds $|X|$. Also, if we encounter a subgroup that is not normal, its quotient Cayley graph does not correspond to a group, but can still be used to obtain a possible contraction.

We are investigating other aspects of group theory that can be applied in this context. We would like to obtain *syntactic characterizations* that enable us to detect whether the communication functions yield a Cayley graph. This will enable us to avoid computation of the cycle notation, and improve the efficiency significantly. In addition, many interesting interconnection networks are themselves based on Cayley graphs that have an underlying group structure [AK89] and we expect this to be useful in the embedding and routing steps of the mapping. We are also studying how to extend this approach to graphs that are "almost" node symmetric.

4.3 Contraction of Arbitrary Task Graphs

Contraction for arbitrary task graphs is achieved by Algorithm MWM-Contract which utilizes a $O(EVlogV)$ maximum weighted matching algorithm to achieve *symmetric contraction*, where E is the number of edges and V the number of vertices in the task graph. By *symmetric contraction* we mean a contraction of the tasks in the task graph into clusters such that the total interprocessor communication is mininimized while satisfiying the load balancing constraint that the total number of tasks per processor be bounded by some constant B.

When 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 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. An example (shown in Fig 5) illustrates the operation of Algorithm MWM-Contract on a system in which 12 tasks must be assigned to 3 processors under the load balancing constraint of at most $B = 4$ tasks per processor. The total IPC = 6 and happens to be optimal in this case, though optimality is not guaranteed.

- 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

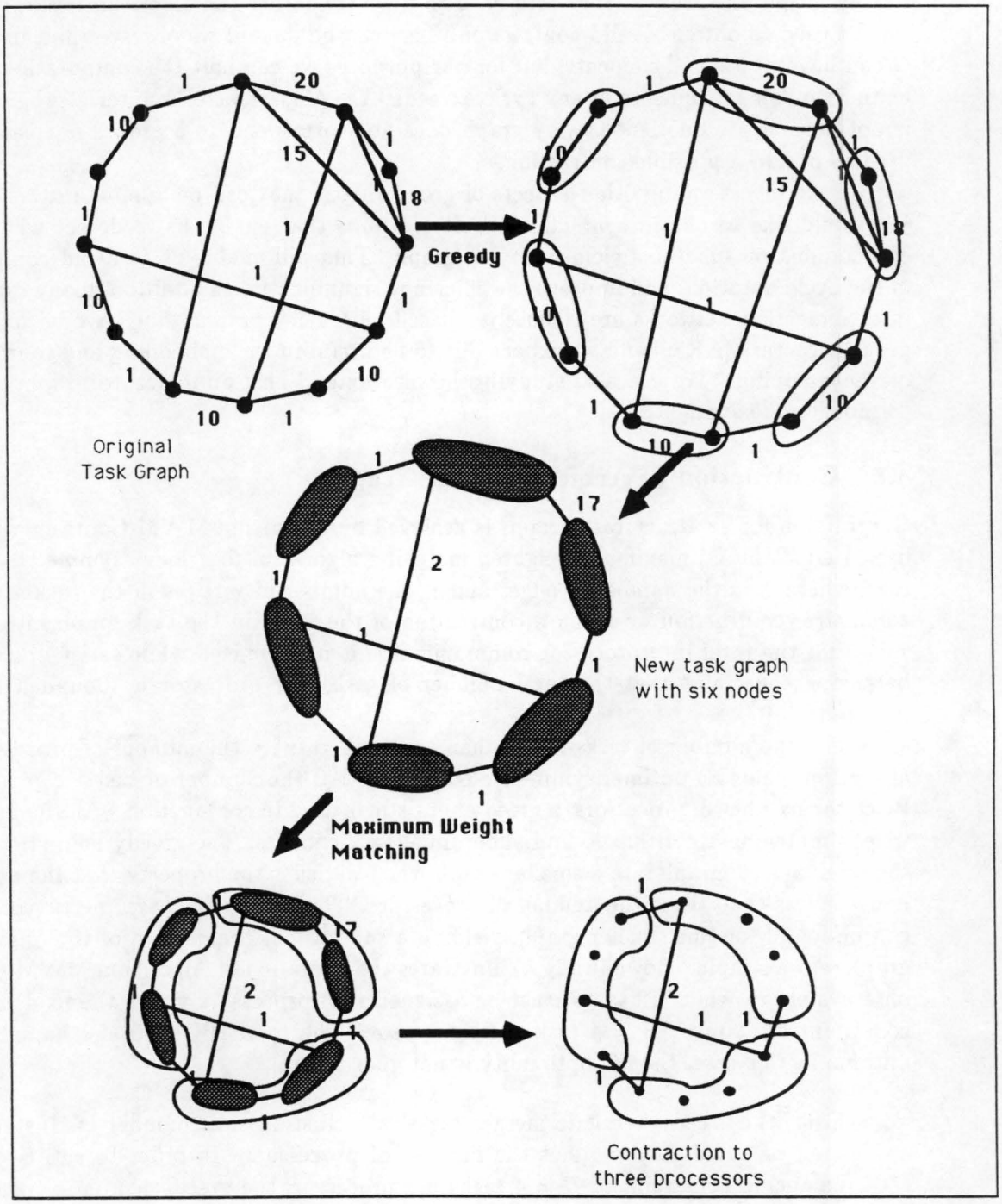


Figure 5: Contraction Example: 12-node task graph to a 3-processor architecture

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 Fig 5b, 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.

Details of Algorithm MWM-Contraction, proofs of optimality, and simulation results can be found in [Lo88]. 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.

4.4 Routing

Routing for both node symmetric and arbitrary task graphs is accomplished using Algorithm MM-Route which uses communication phase information to route messages through the network in a way that minimizes link contention. Given a colored task graph, MM-Route uses a $O(\mid X \mid^2 \mid Y \mid)$ maximal matching algorithm to evenly distribute the edges of a given color to the links of the interconnection network, where X and Y are defined below. Recall that the set of edges of a distinct color represent synchronous communication, i.e. a communication phase. Thus, the heuristic described below is invoked once for each distinct set of colored edges in the task graph.

For example, suppose the *15-body problem* is embedded on an 8-processor hypercube (Fig 6a, where nodes in the task graph are labeled with the LaRCS task numbers, and links in the network are numbered arbitrarily from 1 to 12). We discuss the operation of MM-Route for the chordal edges only; the same procedure would be invoked for the ring edges also.

- 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

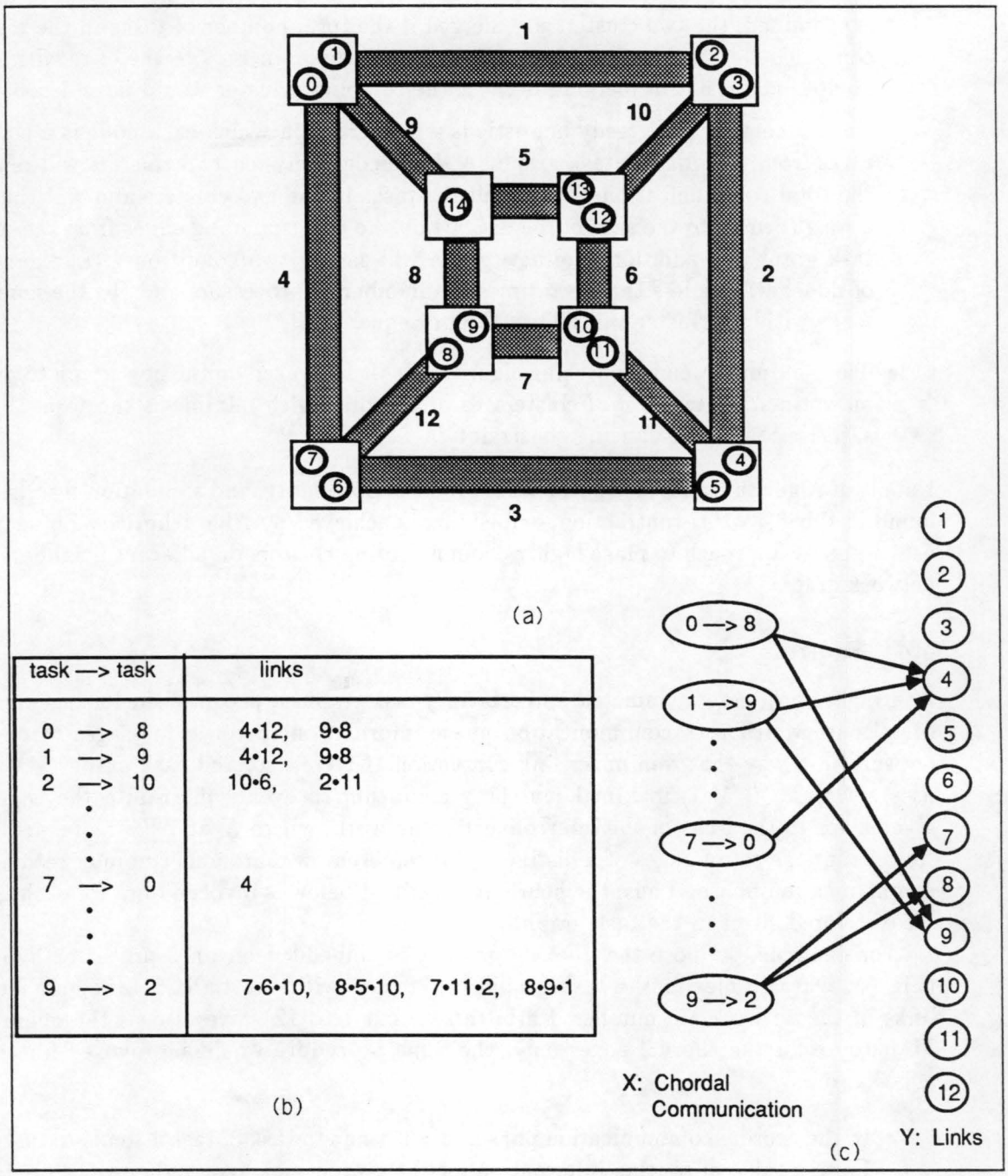


Figure 6: Routing: (a) Embedding of a 15-body problem on an 8-node hypercube, (b) Table of possible routes for the CHORDAL communication step, (c) Bipartite graph for the routing the first hop

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 Fig6c, 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 $M \neq \mid X \mid$ 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 METRICS

METRICS displays the mapping produced automatically by MAPPER, analyzes the mapping according to a spectrum of performance criteria, and allows the user to inspect and modify the mapping. LaRCS and METRICS enrich the power of OREGAMI by teaming user insights and knowledge with the combinatorial optimization capabilities of the group and graph theory algorithms. Modifications to the mapping are made using click and drag mouse operations. The user can reassign tasks to processors or re-route communication edges, and METRICS will display the modified assignment and recompute performance metrics. The performance metrics currently computed by METRICS include: load balancing metrics (tasks per processor, total execution time per processor); link metrics (dilation, volume of communication, communication contention with respect to the phases); and metrics for the overall mapping (completion time of the computation, total interprocessor communication).

6 Ongoing and Future Work

The three main areas for continuing work on the OREGAMI project include scheduling, improved algorithms for MAPPER, and the ability to handle dynamically spawned tasks.

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. 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 [CH74] that specify the allowable ways to multiplex the tasks assigned to a given processor.

Dynamically spawned tasks: OREGAMI currently is designed only for computations in which the number of tasks is static. We wish 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 accomodate dynamically growing parallel computations.

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 wish to investigate include algorithms that perform two or more of the mapping steps simultaneously; algorithms that consider migrating processes at run time in order to accomodate phase shifts (as opposed to our current approach of finding one mapping that accomodates 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 [HF88]). We would like to automatically select the aggregate topology that is 'compatible' with the communication topologies of other phases in the computation. Finally, we will continue to add to the library of 'canned' mappings for nameable task graphs.

The 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 second 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

[AK89] Sheldon B. Akers and Balaji Krishnamurthy. A group-theoretic model for symmetric interconnection networks. *IEEE Transactions on Computers*, 38(4):555–566, April 1989.

[BC86] D. A. Bailey and J. E. Cuny. Graph grammar based specification of interconnection structures for massively parallel computations. In *Graph Grammars and Their Application to Computer Science*, pages 73–85. Springer-Verlag, 1986.

[Ber87] F. Berman. Experience with an automatic solution to the mapping problem. In *The Characteristics of Parallel Algorithms*, pages 307–334. The MIT Press, 1987.

[Bok87] S. H. Bokhari. *Assignment Problems in Parallel and Distributed Computing.* Kluwer Academic Publishers, 1987.

[BS87] F. Berman and L. Snyder. On mapping parallel algorithms into parallel architectures. *Journal of Parallel and Distributed Computing*, 4(5):439–458, October 1987.

[CH74] R. H. Campbell and A. N. Habermann. *The Specification of Process Synchronization by Path Expressions*, volume 16, pages 89–102. Springer-Verlag, 1974.

[Che86] Marina C. Chen. A design methodology for synthesizing parallel algorithms and architectures. *Journal of Parallel and Distributed Computing*, 3(6):461–491, December 1986.

[Cof76] E. G. Coffman. *Computer and Job-Shop Scheduling Theory.* Wiley Interscience, 1976.

[CS84] Peter R. Cappello and Kenneth Steiglitz. Unifying VLSI designs with linear transformations of space-time. *Advances in Computing Research*, 2:23–65, 1984.

[FF82] J. P. Fishburn and R. A. Finkel. Quotient networks. *IEEE Transactions on Computers*, C-31(4):288–295, April 1982.

[FM85] Jose A. B. Fortes and D. Moldovan. Parallelism detection and transformation techniques useful for VLSI algorithms. *Journal of Parallel and Distributed Computing*, 2, August 1985.

[GH88] A. K. Gupta and S.E. Hambrusch. Embedding complete binary trees into butterfly networks. Technical Report CSD-TR-821, Purdue University, 1988.

[HF88] 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.

[HS86] W. D. Hillis and G. L. Steele. Data parallel algorithms. *Communications of the ACM*, 29(12):1170–1183, December 1986.

[Joh87] S. Lennart Johnsson. Data parallel programming and basic linear algebra subroutines. Technical Report YALE/DCS/TR-584, Yale University, September 1987.

[KK] S. M. Kaplan and G. E. Kaiser. Garp: Graph abstractions for concurrent programming. unknown.

[KL80] H. T. Kung and C. E. Leiserson. *Algorithms for VLSI Processor Arrays*, chapter 8.3, pages 271–292. Addison-Wesley, Reading, Ma, 1980.

[Lo88] V. M. Lo. Algorithms for static task assignment and symmetric contraction in distributed computing systems. In *Proceedings of the 1988 International Conference on Parallel Processing*, pages 239–244, August 1988.

[LRG+] V. M. Lo, S. Rajopadhye, S. Gupta, D. Keldsen, M. A. Mohamed, and J. Telle. LaRCS: Language for regular communication structures. in preparation.

[LRG+89] V. M. Lo, S. Rajopadhye, S. Gupta, D. Keldsen, M. A. Mohamed, and J. Telle. Mapping divide-and-conquer algorithms to parallel architectures. Technical Report 89-19, University of Oregon, Dept. of Computer Science, 1989.

[NS87] 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.

[Pol88] C. D. Polychronopoulos. *Parallel Programming and Compilers.* Kluwer Academic Publishers, 1988.

[RF88] Sanjay V. Rajopadhye and Richard M. Fujimoto. Synthesizing systolic arrays from recurrence equations. *Parallel Computing*, page Accepted for Publication, 1988.

[Ros88] A. L. Rosenberg. Graph embeddings 1988: Recent breakthroughs new directions. Technical Report 88-28, University of Massachusetts at Amherst, March 1988.

[Sei85] C. L. Seitz. The cosmic cube. *Communications of the ACM*, 28(1):22–33, January 1985.

[Sto77] H. S. Stone. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Transactions on Software Engineering*, SE-3(1):85–93, January 1977.